Abstract Rewriting

Christian Sternagel and René Thiemann

June 17, 2024

Abstract

We present an Isabelle formalization of abstract rewriting (see, e.g., [1]). First, we define standard relations like *joinability*, *meetability*, *conversion*, etc. Then, we formalize important properties of abstract rewrite systems, e.g., confluence and strong normalization. Our main concern is on strong normalization, since this formalization is the basis of [3] (which is mainly about strong normalization of term rewrite systems; see also IsaFoR/CeTA's website[1]). Hence lemmas involving strong normalization, constitute by far the biggest part of this theory. One of those is Newman's lemma.

Contents

[1] http://cl-informatik.uibk.ac.at/software/ceta

A description of this formalization will be available in [2].

1 Infinite Sequences

theory *Seq*
imports
Main
HOL−Library.Infinite-Set
begin

Infinite sequences are represented by functions of type *nat* ⇒ *'a*.

type-synonym *'a seq* = *nat* ⇒ *'a*

1.1 Operations on Infinite Sequences

An infinite sequence is *linked* by a binary predicate *P* if every two consecutive elements satisfy it. Such a sequence is called a *P-chain*.

abbreviation (*input*) *chainp* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒*'a seq* ⇒ *bool* **where**
chainp P S ≡ ∀ *i*. *P* (*S i*) (*S* (*Suc i*))

Special version for relations.

abbreviation (*input*) *chain* :: *'a rel* ⇒ *'a seq* ⇒ *bool* **where**
chain r S ≡ *chainp* (λ*x y*. (*x*, *y*) ∈ *r*) *S*

Extending a chain at the front.

lemma *cons-chainp*:
assumes *P x* (*S 0*) **and** *chainp P S*
shows *chainp P* (*case-nat x S*) (**is** *chainp P ?S*)
⟨*proof*⟩

Special version for relations.

lemma *cons-chain*:
assumes (*x*, *S 0*) ∈ *r* **and** *chain r S* **shows** *chain r* (*case-nat x S*)
⟨*proof*⟩

A chain admits arbitrary transitive steps.

lemma *chainp-imp-relpowp*:
assumes *chainp P S* **shows** (*P*^^*j*) (*S i*) (*S* (*i* + *j*))
⟨*proof*⟩

lemma *chain-imp-relpow*:
assumes *chain r S* **shows** $(S\ i,\ S\ (i + j)) \in r\widehat{\ }\widehat{\ }j$
⟨*proof*⟩

lemma *chainp-imp-tranclp*:
assumes *chainp P S* **and** $i < j$ **shows** $P\widehat{\ }{++}\ (S\ i)\ (S\ j)$
⟨*proof*⟩

lemma *chain-imp-trancl*:
assumes *chain r S* **and** $i < j$ **shows** $(S\ i,\ S\ j) \in r\widehat{\ }{+}$
⟨*proof*⟩

A chain admits arbitrary reflexive and transitive steps.

lemma *chainp-imp-rtranclp*:
assumes *chainp P S* **and** $i \leq j$ **shows** $P\widehat{\ }{**}\ (S\ i)\ (S\ j)$
⟨*proof*⟩

lemma *chain-imp-rtrancl*:
assumes *chain r S* **and** $i \leq j$ **shows** $(S\ i,\ S\ j) \in r\widehat{\ }{*}$
⟨*proof*⟩

If for every i there is a later index $f\ i$ such that the corresponding elements satisfy the predicate P, then there is a P-chain.

lemma *stepfun-imp-chainp'*:
assumes $\forall\, i{\geq}n{::}nat.\ f\ i \geq i \wedge P\ (S\ i)\ (S\ (f\ i))$
shows *chainp P* $(\lambda i.\ S\ ((f\ \widehat{\ }\widehat{\ }\ i)\ n))$ (**is** *chainp P ?T*)
⟨*proof*⟩

lemma *stepfun-imp-chainp*:
assumes $\forall\, i{\geq}n{::}nat.\ f\ i > i \wedge P\ (S\ i)\ (S\ (f\ i))$
shows *chainp P* $(\lambda i.\ S\ ((f\ \widehat{\ }\widehat{\ }\ i)\ n))$ (**is** *chainp P ?T*)
⟨*proof*⟩

lemma *subchain*:
assumes $\forall\, i{::}nat{>}n.\ \exists j{>}i.\ P\ (f\ i)\ (f\ j)$
shows $\exists \varphi.\ (\forall\, i\ j.\ i < j \longrightarrow \varphi\ i < \varphi\ j) \wedge (\forall\, i.\ P\ (f\ (\varphi\ i))\ (f\ (\varphi\ (Suc\ i))))$
⟨*proof*⟩

If for every i there is a later index j such that the corresponding elements satisfy the predicate P, then there is a P-chain.

lemma *steps-imp-chainp'*:
assumes $\forall\, i{\geq}n{::}nat.\ \exists j{\geq}i.\ P\ (S\ i)\ (S\ j)$ **shows** $\exists\, T.$ *chainp P T*
⟨*proof*⟩

lemma *steps-imp-chainp*:
assumes $\forall\, i{\geq}n{::}nat.\ \exists j{>}i.\ P\ (S\ i)\ (S\ j)$ **shows** $\exists\, T.$ *chainp P T*
⟨*proof*⟩

1.2 Predicates on Natural Numbers

If some property holds for infinitely many natural numbers, obtain an index function that points to these numbers in increasing order.

locale *infinitely-many* =
  fixes $p :: nat \Rightarrow bool$
  assumes *infinite*: $INFM\ j.\ p\ j$
begin

lemma *inf*: $\exists j \geq i.\ p\ j$ ⟨*proof*⟩

fun *index* :: *nat seq* **where**
  $index\ 0 = (LEAST\ n.\ p\ n)$
| $index\ (Suc\ n) = (LEAST\ k.\ p\ k \wedge k > index\ n)$

lemma *index-p*: $p\ (index\ n)$
⟨*proof*⟩

lemma *index-ordered*: $index\ n < index\ (Suc\ n)$
⟨*proof*⟩

lemma *index-not-p-between*:
  assumes *i1*: $index\ n < i$
    and *i2*: $i < index\ (Suc\ n)$
  shows $\neg\ p\ i$
⟨*proof*⟩

lemma *index-ordered-le*:
  assumes $i \leq j$ **shows** $index\ i \leq index\ j$
⟨*proof*⟩

lemma *index-surj*:
  assumes $k \geq index\ l$
  shows $\exists\, i\ j.\ k = index\ i + j \wedge index\ i + j < index\ (Suc\ i)$
⟨*proof*⟩

lemma *index-ordered-less*:
  assumes $i < j$ **shows** $index\ i < index\ j$
⟨*proof*⟩

lemma *index-not-p-start*: **assumes** *i*: $i < index\ 0$ **shows** $\neg\ p\ i$
⟨*proof*⟩

end

1.3 Assembling Infinite Words from Finite Words

Concatenate infinitely many non-empty words to an infinite word.

fun *inf-concat-simple* :: $(nat \Rightarrow nat) \Rightarrow nat \Rightarrow (nat \times nat)$ **where**

inf-concat-simple f 0 = (*0*, *0*)
| *inf-concat-simple f* (*Suc n*) = (
let (*i*, *j*) = *inf-concat-simple f n in*
if Suc j < *f i then* (*i*, *Suc j*)
else (*Suc i*, *0*))

lemma *inf-concat-simple-add*:
assumes *ck*: *inf-concat-simple f k* = (*i*, *j*)
and *jl*: *j* + *l* < *f i*
shows *inf-concat-simple f* (*k* + *l*) = (*i*,*j* + *l*)
⟨*proof*⟩

lemma *inf-concat-simple-surj-zero*: ∃ *k*. *inf-concat-simple f k* = (*i*,*0*)
⟨*proof*⟩

lemma *inf-concat-simple-surj*:
assumes *j* < *f i*
shows ∃ *k*. *inf-concat-simple f k* = (*i*,*j*)
⟨*proof*⟩

lemma *inf-concat-simple-mono*:
assumes *k* ≤ *k′* **shows** *fst* (*inf-concat-simple f k*) ≤ *fst* (*inf-concat-simple f k′*)
⟨*proof*⟩

fun *inf-concat* :: (*nat* ⇒ *nat*) ⇒ *nat* ⇒ *nat* × *nat* **where**
inf-concat n 0 = (*LEAST j*. *n j* > *0*, *0*)
| *inf-concat n* (*Suc k*) = (*let* (*i*, *j*) = *inf-concat n k in* (*if Suc j* < *n i then* (*i*, *Suc j*) *else* (*LEAST i′*. *i′* > *i* ∧ *n i′* > *0*, *0*)))

lemma *inf-concat-bounds*:
assumes *inf*: *INFM i*. *n i* > *0*
and *res*: *inf-concat n k* = (*i*,*j*)
shows *j* < *n i*
⟨*proof*⟩

lemma *inf-concat-add*:
assumes *res*: *inf-concat n k* = (*i*,*j*)
and *j*: *j* + *m* < *n i*
shows *inf-concat n* (*k* + *m*) = (*i*,*j*+*m*)
⟨*proof*⟩

lemma *inf-concat-step*:
assumes *res*: *inf-concat n k* = (*i*,*j*)
and *j*: *Suc* (*j* + *m*) = *n i*
shows *inf-concat n* (*k* + *Suc m*) = (*LEAST i′*. *i′* > *i* ∧ *0* < *n i′*, *0*)
⟨*proof*⟩

lemma *inf-concat-surj-zero*:
assumes $0 < n\ i$
shows $\exists\, k.$ *inf-concat* $n\ k = (i,\ 0)$
⟨*proof*⟩

lemma *inf-concat-surj*:
assumes j: $j < n\ i$
shows $\exists\, k.$ *inf-concat* $n\ k = (i,\ j)$
⟨*proof*⟩

lemma *inf-concat-mono*:
assumes *inf*: *INFM* $i.\ n\ i > 0$
and *resk*: *inf-concat* $n\ k = (i,\ j)$
and *reskp*: *inf-concat* $n\ k' = (i',\ j')$
and *lt*: $i < i'$
shows $k < k'$
⟨*proof*⟩

lemma *inf-concat-Suc*:
assumes *inf*: *INFM* $i.\ n\ i > 0$
and f: $\bigwedge\ i.\ f\ i\ (n\ i) = f\ (Suc\ i)\ 0$
and *resk*: *inf-concat* $n\ k = (i,\ j)$
and *ressk*: *inf-concat* $n\ (Suc\ k) = (i',\ j')$
shows $f\ i'\ j' = f\ i\ (Suc\ j)$
⟨*proof*⟩

end

2 Abstract Rewrite Systems

theory *Abstract-Rewriting*
imports
HOL−Library.Infinite-Set
Regular−Sets.Regexp-Method
Seq
begin

lemma *trancl-mono-set*:
$r \subseteq s \Longrightarrow r^+ \subseteq s^+$
⟨*proof*⟩

lemma *relpow-mono*:
fixes r :: *'a rel*
assumes $r \subseteq r'$ **shows** $r\ \widehat{}\widehat{}\ n \subseteq r'\ \widehat{}\widehat{}\ n$
⟨*proof*⟩

lemma *refl-inv-image*:
refl $R \Longrightarrow$ *refl* (*inv-image* $R\ f$)

⟨*proof*⟩

2.1 Definitions

Two elements are *joinable* (and then have in the joinability relation) w.r.t. A, iff they have a common reduct.

definition *join* :: $'a\ rel \Rightarrow 'a\ rel$ $((\text{-}^{\downarrow})\ [1000]\ 999)$ **where**
$A^{\downarrow} = A^{*}\ O\ (A^{-1})^{*}$

Two elements are *meetable* (and then have in the meetability relation) w.r.t. A, iff they have a common ancestor.

definition *meet* :: $'a\ rel \Rightarrow 'a\ rel$ $((\text{-}^{\uparrow})\ [1000]\ 999)$ **where**
$A^{\uparrow} = (A^{-1})^{*}\ O\ A^{*}$

The *symmetric closure* of a relation allows steps in both directions.

abbreviation *symcl* :: $'a\ rel \Rightarrow 'a\ rel$ $((\text{-}^{\leftrightarrow})\ [1000]\ 999)$ **where**
$A^{\leftrightarrow} \equiv A \cup A^{-1}$

A *conversion* is a (possibly empty) sequence of steps in the symmetric closure.

definition *conversion* :: $'a\ rel \Rightarrow 'a\ rel$ $((\text{-}^{\leftrightarrow *})\ [1000]\ 999)$ **where**
$A^{\leftrightarrow *} = (A^{\leftrightarrow})^{*}$

The set of *normal forms* of an ARS constitutes all the elements that do not have any successors.

definition *NF* :: $'a\ rel \Rightarrow 'a\ set$ **where**
$NF\ A = \{a.\ A\ ``\ \{a\} = \{\}\}$

definition *normalizability* :: $'a\ rel \Rightarrow 'a\ rel$ $((\text{-}^{!})\ [1000]\ 999)$ **where**
$A^{!} = \{(a,\ b).\ (a,\ b) \in A^{*} \wedge b \in NF\ A\}$

notation (*ASCII*)
symcl ((-^<->) [1000] 999) **and**
conversion ((-^<->*) [1000] 999) **and**
normalizability ((-^!) [1000] 999)

lemma *symcl-converse*:
$(A^{\leftrightarrow})^{-1} = A^{\leftrightarrow}$ ⟨*proof*⟩

lemma *symcl-Un*: $(A \cup B)^{\leftrightarrow} = A^{\leftrightarrow} \cup B^{\leftrightarrow}$ ⟨*proof*⟩

lemma *no-step*:
assumes $A\ ``\ \{a\} = \{\}$ **shows** $a \in NF\ A$
⟨*proof*⟩

lemma *joinI*:
$(a,\ c) \in A^{*} \Longrightarrow (b,\ c) \in A^{*} \Longrightarrow (a,\ b) \in A^{\downarrow}$
⟨*proof*⟩

lemma *joinI-left*:
$(a, b) \in A^* \Longrightarrow (a, b) \in A^{\downarrow}$
⟨*proof*⟩

lemma *joinI-right*: $(b, a) \in A^* \Longrightarrow (a, b) \in A^{\downarrow}$
⟨*proof*⟩

lemma *joinE*:
assumes $(a, b) \in A^{\downarrow}$
obtains c **where** $(a, c) \in A^*$ **and** $(b, c) \in A^*$
⟨*proof*⟩

lemma *joinD*:
$(a, b) \in A^{\downarrow} \Longrightarrow \exists c. (a, c) \in A^* \wedge (b, c) \in A^*$
⟨*proof*⟩

lemma *meetI*:
$(a, b) \in A^* \Longrightarrow (a, c) \in A^* \Longrightarrow (b, c) \in A^{\uparrow}$
⟨*proof*⟩

lemma *meetE*:
assumes $(b, c) \in A^{\uparrow}$
obtains a **where** $(a, b) \in A^*$ **and** $(a, c) \in A^*$
⟨*proof*⟩

lemma *meetD*: $(b, c) \in A^{\uparrow} \Longrightarrow \exists a. (a, b) \in A^* \wedge (a, c) \in A^*$
⟨*proof*⟩

lemma *conversionI*: $(a, b) \in (A^{\leftrightarrow})^* \Longrightarrow (a, b) \in A^{\leftrightarrow*}$
⟨*proof*⟩

lemma *conversion-refl* [*simp*]: $(a, a) \in A^{\leftrightarrow*}$
⟨*proof*⟩

lemma *conversionI'*:
assumes $(a, b) \in A^*$ **shows** $(a, b) \in A^{\leftrightarrow*}$
⟨*proof*⟩

lemma *rtrancl-comp-trancl-conv*:
$r^* \ O \ r = r^+$ ⟨*proof*⟩

lemma *trancl-o-refl-is-trancl*:
$r^+ \ O \ r^= = r^+$ ⟨*proof*⟩

lemma *conversionE*:
$(a, b) \in A^{\leftrightarrow*} \Longrightarrow ((a, b) \in (A^{\leftrightarrow})^* \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

Later declarations are tried first for 'proof' and 'rule,' then have the “main” introduction / elimination rules for constants should be declared last.

declare *joinI-left* [*intro*]
declare *joinI-right* [*intro*]
declare *joinI* [*intro*]
declare *joinD* [*dest*]
declare *joinE* [*elim*]

declare *meetI* [*intro*]
declare *meetD* [*dest*]
declare *meetE* [*elim*]

declare *conversionI'* [*intro*]
declare *conversionI* [*intro*]
declare *conversionE* [*elim*]

lemma *conversion-trans*:
$trans\ (A^{\leftrightarrow *})$
⟨*proof*⟩

lemma *conversion-sym*:
$sym\ (A^{\leftrightarrow *})$
⟨*proof*⟩

lemma *conversion-inv*:
$(x, y) \in R^{\leftrightarrow *} \longleftrightarrow (y, x) \in R^{\leftrightarrow *}$
⟨*proof*⟩

lemma *conversion-converse* [*simp*]:
$(A^{\leftrightarrow *})^{-1} = A^{\leftrightarrow *}$
⟨*proof*⟩

lemma *conversion-rtrancl* [*simp*]:
$(A^{\leftrightarrow *})^* = A^{\leftrightarrow *}$
⟨*proof*⟩

lemma *rtrancl-join-join*:
assumes $(a, b) \in A^*$ **and** $(b, c) \in A^{\downarrow}$ **shows** $(a, c) \in A^{\downarrow}$
⟨*proof*⟩

lemma *join-rtrancl-join*:
assumes $(a, b) \in A^{\downarrow}$ **and** $(c, b) \in A^*$ **shows** $(a, c) \in A^{\downarrow}$
⟨*proof*⟩

lemma *NF-I*: $(\bigwedge b.\ (a, b) \notin A) \Longrightarrow a \in NF\ A$ ⟨*proof*⟩

lemma *NF-E*: $a \in NF\ A \Longrightarrow ((a, b) \notin A \Longrightarrow P) \Longrightarrow P$ ⟨*proof*⟩

declare *NF-I* [*intro*]
declare *NF-E* [*elim*]

lemma *NF-no-step*: $a \in NF\ A \Longrightarrow \forall\, b.\ (a,\ b) \notin A$ ⟨*proof*⟩

lemma *NF-anti-mono*:
assumes $A \subseteq B$ **shows** $NF\ B \subseteq NF\ A$
⟨*proof*⟩

lemma *NF-iff-no-step*: $a \in NF\ A = (\forall\, b.\ (a,\ b) \notin A)$ ⟨*proof*⟩

lemma *NF-no-trancl-step*:
assumes $a \in NF\ A$ **shows** $\forall\, b.\ (a,\ b) \notin A^+$
⟨*proof*⟩

lemma *NF-Id-on-fst-image* [*simp*]: $NF\ (Id\text{-}on\ (fst\ `\ A)) = NF\ A$ ⟨*proof*⟩

lemma *fst-image-NF-Id-on* [*simp*]: $fst\ `\ R = Q \Longrightarrow NF\ (Id\text{-}on\ Q) = NF\ R$ ⟨*proof*⟩

lemma *NF-empty* [*simp*]: $NF\ \{\} = UNIV$ ⟨*proof*⟩

lemma *normalizability-I*: $(a,\ b) \in A^* \Longrightarrow b \in NF\ A \Longrightarrow (a,\ b) \in A^!$
⟨*proof*⟩

lemma *normalizability-I'*: $(a,\ b) \in A^* \Longrightarrow (b,\ c) \in A^! \Longrightarrow (a,\ c) \in A^!$
⟨*proof*⟩

lemma *normalizability-E*: $(a,\ b) \in A^! \Longrightarrow ((a,\ b) \in A^* \Longrightarrow b \in NF\ A \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

declare *normalizability-I'* [*intro*]
declare *normalizability-I* [*intro*]
declare *normalizability-E* [*elim*]

2.2 Properties of ARSs

The following properties on (elements of) ARSs are defined: completeness, Church-Rosser property, semi-completeness, strong normalization, unique normal forms, Weak Church-Rosser property, and weak normalization.

definition *CR-on* :: *'a rel* ⇒ *'a set* ⇒ *bool* **where**
$CR\text{-}on\ r\ A \longleftrightarrow (\forall\, a{\in}A.\ \forall\, b\ c.\ (a,\ b) \in r^* \wedge (a,\ c) \in r^* \longrightarrow (b,\ c) \in join\ r)$

abbreviation *CR* :: *'a rel* ⇒ *bool* **where**
$CR\ r \equiv CR\text{-}on\ r\ UNIV$

definition *SN-on* :: *'a rel* ⇒ *'a set* ⇒ *bool* **where**
$SN\text{-}on\ r\ A \longleftrightarrow \neg\ (\exists f.\ f\ 0 \in A \wedge chain\ r\ f)$

abbreviation SN :: $'a\ rel \Rightarrow bool$ **where**
$SN\ r \equiv SN\text{-}on\ r\ UNIV$

Alternative definition of *SN*.

lemma $SN\text{-}def$: $SN\ r = (\forall x.\ SN\text{-}on\ r\ \{x\})$
$\langle proof \rangle$

definition $UNF\text{-}on$ :: $'a\ rel \Rightarrow 'a\ set \Rightarrow bool$ **where**
$UNF\text{-}on\ r\ A \longleftrightarrow (\forall a{\in}A.\ \forall b\ c.\ (a,\ b) \in r^{!} \wedge (a,\ c) \in r^{!} \longrightarrow b = c)$

abbreviation UNF :: $'a\ rel \Rightarrow bool$ **where** $UNF\ r \equiv UNF\text{-}on\ r\ UNIV$

definition $WCR\text{-}on$ :: $'a\ rel \Rightarrow 'a\ set \Rightarrow bool$ **where**
$WCR\text{-}on\ r\ A \longleftrightarrow (\forall a{\in}A.\ \forall b\ c.\ (a,\ b) \in r \wedge (a,\ c) \in r \longrightarrow (b,\ c) \in join\ r)$

abbreviation WCR :: $'a\ rel \Rightarrow bool$ **where** $WCR\ r \equiv WCR\text{-}on\ r\ UNIV$

definition $WN\text{-}on$ :: $'a\ rel \Rightarrow 'a\ set \Rightarrow bool$ **where**
$WN\text{-}on\ r\ A \longleftrightarrow (\forall a{\in}A.\ \exists b.\ (a,\ b) \in r^{!})$

abbreviation WN :: $'a\ rel \Rightarrow bool$ **where**
$WN\ r \equiv WN\text{-}on\ r\ UNIV$

lemmas $CR\text{-}defs = CR\text{-}on\text{-}def$
lemmas $SN\text{-}defs = SN\text{-}on\text{-}def$
lemmas $UNF\text{-}defs = UNF\text{-}on\text{-}def$
lemmas $WCR\text{-}defs = WCR\text{-}on\text{-}def$
lemmas $WN\text{-}defs = WN\text{-}on\text{-}def$

definition $complete\text{-}on$ :: $'a\ rel \Rightarrow 'a\ set \Rightarrow bool$ **where**
$complete\text{-}on\ r\ A \longleftrightarrow SN\text{-}on\ r\ A \wedge CR\text{-}on\ r\ A$

abbreviation $complete$ :: $'a\ rel \Rightarrow bool$ **where**
$complete\ r \equiv complete\text{-}on\ r\ UNIV$

definition $semi\text{-}complete\text{-}on$ :: $'a\ rel \Rightarrow 'a\ set \Rightarrow bool$ **where**
$semi\text{-}complete\text{-}on\ r\ A \longleftrightarrow\ WN\text{-}on\ r\ A \wedge CR\text{-}on\ r\ A$

abbreviation $semi\text{-}complete$ :: $'a\ rel \Rightarrow bool$ **where**
$semi\text{-}complete\ r \equiv semi\text{-}complete\text{-}on\ r\ UNIV$

lemmas $complete\text{-}defs = complete\text{-}on\text{-}def$
lemmas $semi\text{-}complete\text{-}defs = semi\text{-}complete\text{-}on\text{-}def$

Unique normal forms with respect to conversion.

definition UNC :: $'a\ rel \Rightarrow bool$ **where**
$UNC\ A \longleftrightarrow (\forall a\ b.\ a \in NF\ A \wedge b \in NF\ A \wedge (a,\ b) \in A^{\leftrightarrow *} \longrightarrow a = b)$

lemma $complete\text{-}onI$:

SN-on r A $\Longrightarrow$ *CR-on r A* $\Longrightarrow$ *complete-on r A*
⟨*proof*⟩

lemma *complete-onE*:
complete-on r A $\Longrightarrow$ (*SN-on r A* $\Longrightarrow$ *CR-on r A* $\Longrightarrow$ *P*) $\Longrightarrow$ *P*
⟨*proof*⟩

lemma *CR-onI*:
$(\bigwedge a\ b\ c.\ a \in A \Longrightarrow (a, b) \in r^* \Longrightarrow (a, c) \in r^* \Longrightarrow (b, c) \in join\ r) \Longrightarrow$ *CR-on r A*
⟨*proof*⟩

lemma *CR-on-singletonI*:
$(\bigwedge b\ c.\ (a, b) \in r^* \Longrightarrow (a, c) \in r^* \Longrightarrow (b, c) \in join\ r) \Longrightarrow$ *CR-on r* $\{a\}$
⟨*proof*⟩

lemma *CR-onE*:
CR-on r A $\Longrightarrow a \in A \Longrightarrow ((b, c) \in join\ r \Longrightarrow P) \Longrightarrow ((a, b) \notin r^* \Longrightarrow P) \Longrightarrow ((a, c) \notin r^* \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

lemma *CR-onD*:
CR-on r A $\Longrightarrow a \in A \Longrightarrow (a, b) \in r^* \Longrightarrow (a, c) \in r^* \Longrightarrow (b, c) \in join\ r$
⟨*proof*⟩

lemma *semi-complete-onI*: *WN-on r A* $\Longrightarrow$ *CR-on r A* $\Longrightarrow$ *semi-complete-on r A*
⟨*proof*⟩

lemma *semi-complete-onE*:
semi-complete-on r A $\Longrightarrow$ (*WN-on r A* $\Longrightarrow$ *CR-on r A* $\Longrightarrow$ *P*) $\Longrightarrow$ *P*
⟨*proof*⟩

declare *semi-complete-onI* [*intro*]
declare *semi-complete-onE* [*elim*]

declare *complete-onI* [*intro*]
declare *complete-onE* [*elim*]

declare *CR-onI* [*intro*]
declare *CR-on-singletonI* [*intro*]

declare *CR-onD* [*dest*]
declare *CR-onE* [*elim*]

lemma *UNC-I*:
$(\bigwedge a\ b.\ a \in NF\ A \Longrightarrow b \in NF\ A \Longrightarrow (a, b) \in A^{\leftrightarrow *} \Longrightarrow a = b) \Longrightarrow UNC\ A$
⟨*proof*⟩

lemma *UNC-E*:

$\llbracket UNC\ A;\ a = b \Longrightarrow P;\ a \notin NF\ A \Longrightarrow P;\ b \notin NF\ A \Longrightarrow P;\ (a,\ b) \notin A^{\leftrightarrow *} \Longrightarrow P\rrbracket \Longrightarrow P$
⟨*proof*⟩

lemma *UNF-onI*: $(\bigwedge a\ b\ c.\ a \in A \Longrightarrow (a,\ b) \in r^! \Longrightarrow (a,\ c) \in r^! \Longrightarrow b = c) \Longrightarrow UNF\text{-}on\ r\ A$
⟨*proof*⟩

lemma *UNF-onE*:
$UNF\text{-}on\ r\ A \Longrightarrow a \in A \Longrightarrow (b = c \Longrightarrow P) \Longrightarrow ((a,\ b) \notin r^! \Longrightarrow P) \Longrightarrow ((a,\ c) \notin r^! \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

lemma *UNF-onD*:
$UNF\text{-}on\ r\ A \Longrightarrow a \in A \Longrightarrow (a,\ b) \in r^! \Longrightarrow (a,\ c) \in r^! \Longrightarrow b = c$
⟨*proof*⟩

declare *UNF-onI* [*intro*]
declare *UNF-onD* [*dest*]
declare *UNF-onE* [*elim*]

lemma *SN-onI*:
assumes $\bigwedge f.\ \llbracket f\ 0 \in A;\ chain\ r\ f\rrbracket \Longrightarrow False$
shows *SN-on r A*
⟨*proof*⟩

lemma *SN-I*: $(\bigwedge a.\ SN\text{-}on\ A\ \{a\}) \Longrightarrow SN\ A$
⟨*proof*⟩

lemma *SN-on-trancl-imp-SN-on*:
assumes *SN-on* (R^+) *T* **shows** *SN-on R T*
⟨*proof*⟩

lemma *SN-onE*:
assumes *SN-on r A*
and $\neg\ (\exists f.\ f\ 0 \in A \land chain\ r\ f) \Longrightarrow P$
shows *P*
⟨*proof*⟩

lemma *not-SN-onE*:
assumes $\neg$ *SN-on r A*
and $\bigwedge f.\ \llbracket f\ 0 \in A;\ chain\ r\ f\rrbracket \Longrightarrow P$
shows *P*
⟨*proof*⟩

declare *SN-onI* [*intro*]
declare *SN-onE* [*elim*]
declare *not-SN-onE* [*Pure.elim*, *elim*]

lemma *refl-not-SN*: $(x, x) \in R \Longrightarrow \neg$ *SN* R
⟨*proof*⟩

lemma *SN-on-irrefl*:
assumes *SN-on* r A
shows $\forall a \in A.\ (a, a) \notin r$
⟨*proof*⟩

lemma *WCR-onI*: $(\bigwedge a\ b\ c.\ a \in A \Longrightarrow (a, b) \in r \Longrightarrow (a, c) \in r \Longrightarrow (b, c) \in$ *join* $r) \Longrightarrow$ *WCR-on* r A
⟨*proof*⟩

lemma *WCR-onE*:
WCR-on r $A \Longrightarrow a \in A \Longrightarrow ((b, c) \in$ *join* $r \Longrightarrow P) \Longrightarrow ((a, b) \notin r \Longrightarrow P) \Longrightarrow ((a, c) \notin r \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

lemma *SN-nat-bounded*: *SN* $\{(x, y :: nat).\ x < y \wedge y \leq b\}$ (**is** *SN ?R*)
⟨*proof*⟩

lemma *WCR-onD*:
WCR-on r $A \Longrightarrow a \in A \Longrightarrow (a, b) \in r \Longrightarrow (a, c) \in r \Longrightarrow (b, c) \in$ *join* r
⟨*proof*⟩

lemma *WN-onI*: $(\bigwedge a.\ a \in A \Longrightarrow \exists b.\ (a, b) \in r^!) \Longrightarrow$ *WN-on* r A
⟨*proof*⟩

lemma *WN-onE*: *WN-on* r $A \Longrightarrow a \in A \Longrightarrow (\bigwedge b.\ (a, b) \in r^! \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

lemma *WN-onD*: *WN-on* r $A \Longrightarrow a \in A \Longrightarrow \exists b.\ (a, b) \in r^!$
⟨*proof*⟩

declare *WCR-onI* [*intro*]
declare *WCR-onD* [*dest*]
declare *WCR-onE* [*elim*]

declare *WN-onI* [*intro*]
declare *WN-onD* [*dest*]
declare *WN-onE* [*elim*]

Restricting a relation r to those elements that are strongly normalizing with respect to a relation s.

definition *restrict-SN* :: *'a rel* ⇒ *'a rel* ⇒ *'a rel* **where**
restrict-SN r s = $\{(a, b) \mid a\ b.\ (a, b) \in r \wedge$ *SN-on* s $\{a\}\}$

lemma *SN-restrict-SN-idemp* [*simp*]: *SN* (*restrict-SN* A A)
⟨*proof*⟩

lemma *SN-on-Image*:
assumes *SN-on r A*
shows *SN-on r* (*r* ‘‘ *A*)
⟨*proof*⟩

lemma *SN-on-subset2*:
assumes $A \subseteq B$ **and** *SN-on r B*
shows *SN-on r A*
⟨*proof*⟩

lemma *step-preserves-SN-on*:
assumes *1*: $(a, b) \in r$
and *2*: *SN-on r* $\{a\}$
shows *SN-on r* $\{b\}$
⟨*proof*⟩

lemma *steps-preserve-SN-on*: $(a, b) \in A^* \Longrightarrow$ *SN-on A* $\{a\} \Longrightarrow$ *SN-on A* $\{b\}$
⟨*proof*⟩

lemma *relpow-seq*:
assumes $(x, y) \in r\^\^n$
shows $\exists f.\ f\ 0 = x \wedge f\ n = y \wedge (\forall i<n.\ (f\ i,\ f\ (Suc\ i)) \in r)$
⟨*proof*⟩

lemma *rtrancl-imp-seq*:
assumes $(x, y) \in r^*$
shows $\exists f\ n.\ f\ 0 = x \wedge f\ n = y \wedge (\forall i<n.\ (f\ i,\ f\ (Suc\ i)) \in r)$
⟨*proof*⟩

lemma *SN-on-Image-rtrancl*:
assumes *SN-on r A*
shows *SN-on r* (r^* ‘‘ *A*)
⟨*proof*⟩

declare *subrelI* [*Pure.intro*]

lemma *restrict-SN-trancl-simp* [*simp*]: (*restrict-SN A A*)$^+$ = *restrict-SN* (A^+) *A*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *SN-imp-WN*:
assumes *SN A* **shows** *WN A*
⟨*proof*⟩

lemma *UNC-imp-UNF*:
assumes *UNC r* **shows** *UNF r*
⟨*proof*⟩

lemma *join-NF-imp-eq*:
assumes $(x, y) \in r^{\downarrow}$ **and** $x \in NF\ r$ **and** $y \in NF\ r$
shows $x = y$
⟨*proof*⟩

lemma *rtrancl-Restr*:
assumes $(x, y) \in (Restr\ r\ A)^*$
shows $(x, y) \in r^*$
⟨*proof*⟩

lemma *join-mono*:
assumes $r \subseteq s$
shows $r^{\downarrow} \subseteq s^{\downarrow}$
⟨*proof*⟩

lemma *CR-iff-meet-subset-join*: $CR\ r = (r^{\uparrow} \subseteq r^{\downarrow})$
⟨*proof*⟩

lemma *CR-divergence-imp-join*:
assumes $CR\ r$ **and** $(x, y) \in r^*$ **and** $(x, z) \in r^*$
shows $(y, z) \in r^{\downarrow}$
⟨*proof*⟩

lemma *join-imp-conversion*: $r^{\downarrow} \subseteq r^{\leftrightarrow *}$
⟨*proof*⟩

lemma *meet-imp-conversion*: $r^{\uparrow} \subseteq r^{\leftrightarrow *}$
⟨*proof*⟩

lemma *CR-imp-UNF*:
assumes $CR\ r$ **shows** $UNF\ r$
⟨*proof*⟩

lemma *CR-iff-conversion-imp-join*: $CR\ r = (r^{\leftrightarrow *} \subseteq r^{\downarrow})$
⟨*proof*⟩

lemma *CR-imp-conversionIff-join*:
assumes $CR\ r$ **shows** $r^{\leftrightarrow *} = r^{\downarrow}$
⟨*proof*⟩

lemma *sym-join*: $sym\ (join\ r)$ ⟨*proof*⟩

lemma *join-sym*: $(s, t) \in A^{\downarrow} \Longrightarrow (t, s) \in A^{\downarrow}$ ⟨*proof*⟩

lemma *CR-join-left-I*:
assumes $CR\ r$ **and** $(x, y) \in r^*$ **and** $(x, z) \in r^{\downarrow}$ **shows** $(y, z) \in r^{\downarrow}$
⟨*proof*⟩

lemma *CR-join-right-I*:
assumes *CR r* **and** $(x, y) \in r^{\downarrow}$ **and** $(y, z) \in r^*$ **shows** $(x, z) \in r^{\downarrow}$
⟨*proof*⟩

lemma *NF-not-suc*:
assumes $(x, y) \in r^*$ **and** $x \in NF\ r$ **shows** $x = y$
⟨*proof*⟩

lemma *semi-complete-imp-conversionIff-same-NF*:
assumes *semi-complete r*
shows $((x, y) \in r^{\leftrightarrow *}) = (\forall u\ v.\ (x, u) \in r^! \wedge (y, v) \in r^! \longrightarrow u = v)$
⟨*proof*⟩

lemma *CR-imp-UNC*:
assumes *CR r* **shows** *UNC r*
⟨*proof*⟩

lemma *WN-UNF-imp-CR*:
assumes *WN r* **and** *UNF r* **shows** *CR r*
⟨*proof*⟩

definition *diamond* :: *'a rel* ⇒ *bool* (◊) **where**
$\Diamond\ r \longleftrightarrow (r^{-1}\ O\ r) \subseteq (r\ O\ r^{-1})$

lemma *diamond-I* [*intro*]: $(r^{-1}\ O\ r) \subseteq (r\ O\ r^{-1}) \Longrightarrow \Diamond\ r$ ⟨*proof*⟩

lemma *diamond-E* [*elim*]: $\Diamond\ r \Longrightarrow ((r^{-1}\ O\ r) \subseteq (r\ O\ r^{-1}) \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

lemma *diamond-imp-semi-confluence*:
assumes $\Diamond\ r$ **shows** $(r^{-1}\ O\ r^*) \subseteq r^{\downarrow}$
⟨*proof*⟩

lemma *semi-confluence-imp-CR*:
assumes $(r^{-1}\ O\ r^*) \subseteq r^{\downarrow}$ **shows** *CR r*
⟨*proof*⟩

lemma *diamond-imp-CR*:
assumes $\Diamond\ r$ **shows** *CR r*
⟨*proof*⟩

lemma *diamond-imp-CR'*:
assumes $\Diamond\ s$ **and** $r \subseteq s$ **and** $s \subseteq r^*$ **shows** *CR r*
⟨*proof*⟩

lemma *SN-imp-minimal*:
assumes *SN A*
shows $\forall\ Q\ x.\ x \in Q \longrightarrow (\exists\ z{\in}Q.\ \forall\ y.\ (z, y) \in A \longrightarrow y \notin Q)$

⟨*proof*⟩

lemma *SN-on-imp-on-minimal*:
assumes *SN-on r* $\{x\}$
shows $\forall Q.\ x \in Q \longrightarrow (\exists z{\in}Q.\ \forall y.\ (z, y) \in r \longrightarrow y \notin Q)$
⟨*proof*⟩

lemma *minimal-imp-wf*:
assumes $\forall Q\ x.\ x \in Q \longrightarrow (\exists z{\in}Q.\ \forall y.\ (z, y) \in r \longrightarrow y \notin Q)$
shows $wf(r^{-1})$
⟨*proof*⟩

lemmas *SN-imp-wf* = *SN-imp-minimal* [*THEN minimal-imp-wf*]

lemma *wf-imp-SN*:
assumes $wf\ (A^{-1})$ **shows** *SN A*
⟨*proof*⟩

lemma *SN-nat-gt*: *SN* $\{(a, b :: nat)\ .\ a > b\}$
⟨*proof*⟩

lemma *SN-iff-wf*: $SN\ A = wf\ (A^{-1})$ ⟨*proof*⟩

lemma *SN-imp-acyclic*: $SN\ R \Longrightarrow acyclic\ R$
⟨*proof*⟩

lemma *SN-induct*:
assumes *sn*: *SN r* **and** *step*: $\bigwedge a.\ (\bigwedge b.\ (a, b) \in r \Longrightarrow P\ b) \Longrightarrow P\ a$
shows *P a*
⟨*proof*⟩

lemmas *SN-induct-rule* = *SN-induct* [*consumes 1*, *case-names IH*, *induct pred*: *SN*]

lemma *SN-on-induct* [*consumes 2*, *case-names IH*, *induct pred*: *SN-on*]:
assumes *SN*: *SN-on R A*
and $s \in A$
and *imp*: $\bigwedge t.\ (\bigwedge u.\ (t, u) \in R \Longrightarrow P\ u) \Longrightarrow P\ t$
shows *P s*
⟨*proof*⟩

lemma *accp-imp-SN-on*:
assumes $\bigwedge x.\ x \in A \Longrightarrow Wellfounded.accp\ g\ x$
shows *SN-on* $\{(y, z).\ g\ z\ y\}$ *A*
⟨*proof*⟩

lemma *SN-on-imp-accp*:
assumes *SN-on* $\{(y, z).\ g\ z\ y\}\ A$
shows $\forall x \in A.$ *Wellfounded.accp* $g\ x$
⟨*proof*⟩

lemma *SN-on-conv-accp*:
SN-on $\{(y, z).\ g\ z\ y\}\ \{x\} =$ *Wellfounded.accp* $g\ x$
⟨*proof*⟩

lemma *SN-on-conv-acc*: *SN-on* $\{(y, z).\ (z, y) \in r\}\ \{x\} \longleftrightarrow x \in$ *Wellfounded.acc* r
⟨*proof*⟩

lemma *acc-imp-SN-on*:
assumes $x \in$ *Wellfounded.acc* r **shows** *SN-on* $\{(y, z).\ (z, y) \in r\}\ \{x\}$
⟨*proof*⟩

lemma *SN-on-imp-acc*:
assumes *SN-on* $\{(y, z).\ (z, y) \in r\}\ \{x\}$ **shows** $x \in$ *Wellfounded.acc* r
⟨*proof*⟩

2.3 Newman's Lemma

lemma *rtrancl-len-E* [*elim*]:
assumes $(x, y) \in r^*$ **obtains** n **where** $(x, y) \in r^{\frown\frown}n$
⟨*proof*⟩

lemma *relpow-Suc-E2'* [*elim*]:
assumes $(x, z) \in A^{\frown\frown}Suc\ n$ **obtains** y **where** $(x, y) \in A$ **and** $(y, z) \in A^*$
⟨*proof*⟩

lemmas *SN-on-induct'* [*consumes 1*, *case-names IH*] = *SN-on-induct* [*OF - singletonI*]

lemma *Newman-local*:
assumes *SN-on* $r\ X$ **and** *WCR*: *WCR-on* $r\ \{x.$ *SN-on* $r\ \{x\}\}$
shows *CR-on* $r\ X$
⟨*proof*⟩

lemma *Newman*: *SN* $r \Longrightarrow$ *WCR* $r \Longrightarrow$ *CR* r
⟨*proof*⟩

lemma *Image-SN-on*:
assumes *SN-on* $r\ (r\ ``\ A)$
shows *SN-on* $r\ A$
⟨*proof*⟩

lemma *SN-on-Image-conv*: *SN-on* $r\ (r\ ``\ A) =$ *SN-on* $r\ A$
⟨*proof*⟩

If all successors are terminating, then the current element is also terminating.

lemma *step-reflects-SN-on*:
assumes $(\bigwedge b.\ (a,\ b) \in r \Longrightarrow \mathit{SN\text{-}on}\ r\ \{b\})$
shows $\mathit{SN\text{-}on}\ r\ \{a\}$
⟨*proof*⟩

lemma *SN-on-all-reducts-SN-on-conv*:
$\mathit{SN\text{-}on}\ r\ \{a\} = (\forall\, b.\ (a,\ b) \in r \longrightarrow \mathit{SN\text{-}on}\ r\ \{b\})$
⟨*proof*⟩

lemma *SN-imp-SN-trancl*: $\mathit{SN}\ R \Longrightarrow \mathit{SN}\ (R^+)$
⟨*proof*⟩

lemma *SN-trancl-imp-SN*:
assumes $\mathit{SN}\ (R^+)$ **shows** $\mathit{SN}\ R$
⟨*proof*⟩

lemma *SN-trancl-SN-conv*: $\mathit{SN}\ (R^+) = \mathit{SN}\ R$
⟨*proof*⟩

lemma *SN-inv-image*: $\mathit{SN}\ R \Longrightarrow \mathit{SN}\ (\mathit{inv\text{-}image}\ R\ f)$ ⟨*proof*⟩

lemma *SN-subset*: $\mathit{SN}\ R \Longrightarrow R' \subseteq R \Longrightarrow \mathit{SN}\ R'$ ⟨*proof*⟩

lemma *SN-pow-imp-SN*:
assumes $\mathit{SN}\ (A\widehat{\ \ }\mathit{Suc}\ n)$ **shows** $\mathit{SN}\ A$
⟨*proof*⟩

lemma *pow-Suc-subset-trancl*: $R\widehat{\ \ }(\mathit{Suc}\ n) \subseteq R^+$
⟨*proof*⟩

lemma *SN-imp-SN-pow*:
assumes $\mathit{SN}\ R$ **shows** $\mathit{SN}\ (R\widehat{\ \ }\mathit{Suc}\ n)$
⟨*proof*⟩

lemma *SN-pow*: $\mathit{SN}\ R \longleftrightarrow \mathit{SN}\ (R\ \widehat{\ \ }\ \mathit{Suc}\ n)$
⟨*proof*⟩

lemma *SN-on-trancl*:
assumes $\mathit{SN\text{-}on}\ r\ A$ **shows** $\mathit{SN\text{-}on}\ (r^+)\ A$
⟨*proof*⟩

lemma *SN-on-trancl-SN-on-conv*: $\mathit{SN\text{-}on}\ (R^+)\ T = \mathit{SN\text{-}on}\ R\ T$
⟨*proof*⟩

Restrict an ARS to elements of a given set.

definition *restrict* :: $'a\ rel \Rightarrow 'a\ set \Rightarrow 'a\ rel$ **where**
restrict $r\ S = \{(x, y).\ x \in S \wedge y \in S \wedge (x, y) \in r\}$

lemma *SN-on-restrict*:
assumes *SN-on* $r\ A$
shows *SN-on* (*restrict* $r\ S$) A (**is** *SN-on* $?r\ A$)
⟨*proof*⟩

lemma *restrict-rtrancl*: $(restrict\ r\ S)^* \subseteq r^*$ (**is** $?r^* \subseteq r^*$)
⟨*proof*⟩

lemma *rtrancl-Image-step*:
assumes $a \in r^*$ `` A
and $(a, b) \in r^*$
shows $b \in r^*$ `` A
⟨*proof*⟩

lemma *WCR-SN-on-imp-CR-on*:
assumes *WCR* r **and** *SN-on* $r\ A$ **shows** *CR-on* $r\ A$
⟨*proof*⟩

lemma *SN-on-Image-normalizable*:
assumes *SN-on* $r\ A$
shows $\forall a{\in}A.\ \exists b.\ b \in r^!$ `` A
⟨*proof*⟩

lemma *SN-on-imp-normalizability*:
assumes *SN-on* $r\ \{a\}$ **shows** $\exists b.\ (a, b) \in r^!$
⟨*proof*⟩

2.4 Commutation

definition *commute* :: $'a\ rel \Rightarrow 'a\ rel \Rightarrow bool$ **where**
commute $r\ s \longleftrightarrow ((r^{-1})^*\ O\ s^*) \subseteq (s^*\ O\ (r^{-1})^*)$

lemma *CR-iff-self-commute*: $CR\ r = commute\ r\ r$
⟨*proof*⟩

lemma *rtrancl-imp-rtrancl-UN*:
assumes $(x, y) \in r^*$ **and** $r \in I$
shows $(x, y) \in (\bigcup r{\in}I.\ r)^*$ (**is** $(x, y) \in ?r^*$)
⟨*proof*⟩

definition *quasi-commute* :: $'a\ rel \Rightarrow 'a\ rel \Rightarrow bool$ **where**
quasi-commute $r\ s \longleftrightarrow (s\ O\ r) \subseteq r\ O\ (r \cup s)^*$

lemma *rtrancl-union-subset-rtrancl-union-trancl*: $(r \cup s^+)^* = (r \cup s)^*$
⟨*proof*⟩

lemma *qc-imp-qc-trancl*:
assumes *quasi-commute r s* **shows** *quasi-commute r* (s^+)
⟨*proof*⟩

lemma *steps-reflect-SN-on*:
assumes ¬ *SN-on r* $\{b\}$ **and** $(a, b) \in r^*$
shows ¬ *SN-on r* $\{a\}$
⟨*proof*⟩

lemma *chain-imp-not-SN-on*:
assumes *chain r f*
shows ¬ *SN-on r* $\{f\ i\}$
⟨*proof*⟩

lemma *quasi-commute-imp-SN*:
assumes *SN r* **and** *SN s* **and** *quasi-commute r s*
shows *SN* $(r \cup s)$
⟨*proof*⟩

2.5 Strong Normalization

lemma *non-strict-into-strict*:
assumes *compat*: $NS\ O\ S \subseteq S$
and *steps*: $(s, t) \in (NS^*)\ O\ S$
shows $(s, t) \in S$
⟨*proof*⟩

lemma *comp-trancl*:
assumes $R\ O\ S \subseteq S$ **shows** $R\ O\ S^+ \subseteq S^+$
⟨*proof*⟩

lemma *comp-rtrancl-trancl*:
assumes *comp*: $R\ O\ S \subseteq S$
and *seq*: $(s, t) \in (R \cup S)^*\ O\ S$
shows $(s, t) \in S^+$
⟨*proof*⟩

lemma *trancl-union-right*: $r^+ \subseteq (s \cup r)^+$
⟨*proof*⟩

lemma *restrict-SN-subset*: *restrict-SN* $R\ S \subseteq R$
⟨*proof*⟩

lemma *chain-Un-SN-on-imp-first-step*:
assumes *chain* $(R \cup S)$ t **and** *SN-on* S $\{t\ 0\}$
shows $\exists i.\ (t\ i,\ t\ (Suc\ i)) \in R \wedge (\forall j<i.\ (t\ j,\ t\ (Suc\ j)) \in S \wedge (t\ j,\ t\ (Suc\ j)) \notin R)$
⟨*proof*⟩

lemma *first-step*:
assumes *C*: $C = A \cup B$ **and** *steps*: $(x, y) \in C^*$ **and** *Bstep*: $(y, z) \in B$
shows $\exists y. (x, y) \in A^* \ O\ B$
⟨*proof*⟩

lemma *first-step-O*:
assumes *C*: $C = A \cup B$ **and** *steps*: $(x, y) \in C^* \ O\ B$
shows $\exists\ y. (x, y) \in A^* \ O\ B$
⟨*proof*⟩

lemma *firstStep*:
assumes *LSR*: $L = S \cup R$ **and** *xyL*: $(x, y) \in L^*$
shows $(x, y) \in R^* \vee (x, y) \in R^* \ O\ S\ O\ L^*$
⟨*proof*⟩

lemma *non-strict-ending*:
assumes *chain*: *chain* $(R \cup S)\ t$
and *comp*: $R\ O\ S \subseteq S$
and *SN*: *SN-on* $S\ \{t\ 0\}$
shows $\exists j. \forall i \geq j. (t\ i, t\ (Suc\ i)) \in R - S$
⟨*proof*⟩

lemma *SN-on-subset1*:
assumes *SN-on* $r\ A$ **and** $s \subseteq r$
shows *SN-on* $s\ A$
⟨*proof*⟩

lemmas *SN-on-mono* = *SN-on-subset1*

lemma *rtrancl-fun-conv*:
$((s, t) \in R^*) = (\exists\ f\ n.\ f\ 0 = s \wedge f\ n = t \wedge (\forall\ i < n. (f\ i, f\ (Suc\ i)) \in R))$
⟨*proof*⟩

lemma *compat-tr-compat*:
assumes $NS\ O\ S \subseteq S$ **shows** $NS^* \ O\ S \subseteq S$
⟨*proof*⟩

lemma *right-comp-S* [*simp*]:
assumes $(x, y) \in S\ O\ (S\ O\ S^* \ O\ NS^* \cup NS^*)$
shows $(x, y) \in (S\ O\ S^* \ O\ NS^*)$
⟨*proof*⟩

lemma *compatible-SN*:
assumes *SN*: *SN S*
and *compat*: $NS\ O\ S \subseteq S$
shows *SN* $(S\ O\ S^* \ O\ NS^*)$ (**is** *SN ?A*)
⟨*proof*⟩

lemma *compatible-rtrancl-split*:
assumes *compat*: $NS\ O\ S \subseteq S$
and *steps*: $(x, y) \in (NS \cup S)^*$
shows $(x, y) \in S\ O\ S^*\ O\ NS^* \cup NS^*$
⟨*proof*⟩

lemma *compatible-conv*:
assumes *compat*: $NS\ O\ S \subseteq S$
shows $(NS \cup S)^*\ O\ S\ O\ (NS \cup S)^* = S\ O\ S^*\ O\ NS^*$
⟨*proof*⟩

lemma *compatible-SN'*:
assumes *compat*: $NS\ O\ S \subseteq S$ **and** *SN*: *SN S*
shows $SN((NS \cup S)^*\ O\ S\ O\ (NS \cup S)^*)$
⟨*proof*⟩

lemma *rtrancl-diff-decomp*:
assumes $(x, y) \in A^* - B^*$
shows $(x, y) \in A^*\ O\ (A - B)\ O\ A^*$
⟨*proof*⟩

lemma *SN-empty* [*simp*]: *SN* {} ⟨*proof*⟩

lemma *SN-on-weakening*:
assumes *SN-on R1 A*
shows *SN-on* $(R1 \cap R2)$ *A*
⟨*proof*⟩

definition *ideriv* :: $'a\ rel \Rightarrow 'a\ rel \Rightarrow (nat \Rightarrow 'a) \Rightarrow bool$ **where**
ideriv R S as $\longleftrightarrow$ $(\forall\, i.\ (as\ i,\ as\ (Suc\ i)) \in R \cup S) \wedge (INFM\ i.\ (as\ i,\ as\ (Suc\ i)) \in R)$

lemma *ideriv-mono*: $R \subseteq R' \Longrightarrow S \subseteq S' \Longrightarrow ideriv\ R\ S\ as \Longrightarrow ideriv\ R'\ S'\ as$
⟨*proof*⟩

fun
shift :: $(nat \Rightarrow 'a) \Rightarrow nat \Rightarrow nat \Rightarrow 'a$
where
shift f j $= (\lambda\ i.\ f\ (i+j))$

lemma *ideriv-split*:
assumes *ideriv*: *ideriv R S as*
and *nideriv*: $\neg$ *ideriv* $(D \cap (R \cup S))\ (R \cup S - D)$ *as*
shows $\exists\ i.$ *ideriv* $(R - D)\ (S - D)$ (*shift as i*)
⟨*proof*⟩

lemma *ideriv-SN*:

assumes *SN*: *SN S*
and *compat*: *NS O S* $\subseteq$ *S*
and *R*: *R* $\subseteq$ *NS* $\cup$ *S*
shows $\neg$ *ideriv* (*S* $\cap$ *R*) (*R* $-$ *S*) *as*
⟨*proof*⟩

lemma *Infm-shift*: (*INFM i. P* (*shift f n i*)) = (*INFM i. P* (*f i*)) (**is** *?S* = *?O*)
⟨*proof*⟩

lemma *rtrancl-list-conv*:
$(s, t) \in R^* \longleftrightarrow$
($\exists$ *ts. last* (*s* # *ts*) = *t* $\wedge$ ($\forall$ *i*<*length ts.* ((*s* # *ts*) ! *i*, (*s* # *ts*) ! *Suc i*) $\in$ *R*))
(**is** *?l* = *?r*)
⟨*proof*⟩

lemma *SN-reaches-NF*:
assumes *SN-on r* {*x*}
shows $\exists y.\ (x, y) \in r^* \wedge y \in$ *NF r*
⟨*proof*⟩

lemma *SN-WCR-reaches-NF*:
assumes *SN*: *SN-on r* {*x*}
and *WCR*: *WCR-on r* {*x. SN-on r* {*x*}}
shows $\exists!\ y.\ (x, y) \in r^* \wedge y \in$ *NF r*
⟨*proof*⟩

definition *some-NF* :: *'a rel* $\Rightarrow$ *'a* $\Rightarrow$ *'a* **where**
some-NF r x = (*SOME y.* $(x, y) \in r^* \wedge y \in$ *NF r*)

lemma *some-NF*:
assumes *SN*: *SN-on r* {*x*}
shows (*x*, *some-NF r x*) $\in r^* \wedge$ *some-NF r x* $\in$ *NF r*
⟨*proof*⟩

lemma *some-NF-WCR*:
assumes *SN*: *SN-on r* {*x*}
and *WCR*: *WCR-on r* {*x. SN-on r* {*x*}}
and *steps*: $(x, y) \in r^*$
and *NF*: *y* $\in$ *NF r*
shows *y* = *some-NF r x*
⟨*proof*⟩

lemma *some-NF-UNF*:
assumes *UNF*: *UNF r*
and *steps*: $(x, y) \in r^*$
and *NF*: *y* $\in$ *NF r*
shows *y* = *some-NF r x*
⟨*proof*⟩

definition *the-NF* $A\ a = (\mathit{THE}\ b.\ (a,\ b) \in A^!)$

context
  fixes A
  assumes *SN*: *SN* A **and** *CR*: *CR* A
begin
lemma *the-NF*: $(a,\ \textit{the-NF}\ A\ a) \in A^!$
⟨*proof*⟩

lemma *the-NF-NF*: *the-NF* $A\ a \in$ *NF* A
  ⟨*proof*⟩

lemma *the-NF-step*:
  assumes $(a,\ b) \in A$
  shows *the-NF* $A\ a =$ *the-NF* $A\ b$
  ⟨*proof*⟩

lemma *the-NF-steps*:
  assumes $(a,\ b) \in A^*$
  shows *the-NF* $A\ a =$ *the-NF* $A\ b$
  ⟨*proof*⟩

lemma *the-NF-conv*:
  assumes $(a,\ b) \in A^{\leftrightarrow *}$
  shows *the-NF* $A\ a =$ *the-NF* $A\ b$
  ⟨*proof*⟩

end

definition *weak-diamond* :: $'a\ \mathit{rel} \Rightarrow \mathit{bool}$ $(w\Diamond)$ **where**
  $w\Diamond\ r \longleftrightarrow (r^{-1}\ O\ r) - \mathit{Id} \subseteq (r\ O\ r^{-1})$

lemma *weak-diamond-imp-CR*:
  assumes *wd*: $w\Diamond\ r$
  shows *CR* r
⟨*proof*⟩

lemma *steps-imp-not-SN-on*:
  fixes $t :: 'a \Rightarrow 'b$
    and $R :: 'b\ \mathit{rel}$
  assumes *steps*: $\bigwedge x.\ (t\ x,\ t\ (f\ x)) \in R$
  shows $\neg$ *SN-on* $R\ \{t\ x\}$
⟨*proof*⟩

lemma *steps-imp-not-SN*:
  fixes $t :: 'a \Rightarrow 'b$
    and $R :: 'b\ \mathit{rel}$
  assumes *steps*: $\bigwedge x.\ (t\ x,\ t\ (f\ x)) \in R$

shows $\neg$ *SN R*
⟨*proof*⟩

lemma *steps-map*:
assumes *fg*: $\bigwedge t\ u\ R\ .\ P\ t \Longrightarrow Q\ R \Longrightarrow (t,\ u) \in R \Longrightarrow P\ u \wedge (f\ t,\ f\ u) \in g\ R$
and *t*: $P\ t$
and *R*: $Q\ R$
and *S*: $Q\ S$
shows $((t,\ u) \in R^* \longrightarrow (f\ t,\ f\ u) \in (g\ R)^*)$
$\wedge\ ((t,\ u) \in R^*\ \ O\ S\ O\ R^* \longrightarrow (f\ t,\ f\ u) \in (g\ R)^*\ \ O\ (g\ S)\ \ O\ (g\ R)^*)$
⟨*proof*⟩

2.6 Terminating part of a relation

inductive-set
SN-part :: *'a rel* ⇒ *'a set*
for *r* :: *'a rel*
where
SN-partI: $(\bigwedge y.\ (x,\ y) \in r \Longrightarrow y \in \textit{SN-part}\ r) \Longrightarrow x \in \textit{SN-part}\ r$

The accessible part of a relation is the same as the terminating part (just two names for the same definition – modulo argument order). See $(\bigwedge y.\ (y,\ ?x) \in\ ?r \Longrightarrow y \in \textit{Wellfounded.acc}\ ?r) \Longrightarrow\ ?x \in \textit{Wellfounded.acc}\ ?r$.

Characterization of *SN-on* via terminating part.

lemma *SN-on-SN-part-conv*:
SN-on r A $\longleftrightarrow$ $A \subseteq$ *SN-part r*
⟨*proof*⟩

Special case for "full" termination.

lemma *SN-SN-part-UNIV-conv*:
SN r $\longleftrightarrow$ *SN-part r* = *UNIV*
⟨*proof*⟩

lemma *closed-imp-rtrancl-closed*: **assumes** *L*: $L \subseteq A$
and *R*: $R\ ``\ A \subseteq A$
shows $\{t \mid s.\ s \in L \wedge (s,t) \in R\hat{}*\} \subseteq A$
⟨*proof*⟩

lemma *trancl-steps-relpow*: **assumes** $a \subseteq b\hat{}+$
shows $(x,y) \in a\hat{}\hat{}n \Longrightarrow \exists\ m.\ m \geq n \wedge (x,y) \in b\hat{}\hat{}m$
⟨*proof*⟩

lemma *relpow-image*: **assumes** *f*: $\bigwedge\ s\ t.\ (s,t) \in r \Longrightarrow (f\ s,\ f\ t) \in r'$
shows $(s,t) \in r\ \hat{}\hat{}\ n \Longrightarrow (f\ s,\ f\ t) \in r'\ \hat{}\hat{}\ n$
⟨*proof*⟩

lemma *relpow-refl-mono*:
assumes *refl*: $\bigwedge\ x.\ (x,x) \in \textit{Rel}$
shows $m \leq n \Longrightarrow (a,b) \in \textit{Rel}\ \hat{}\hat{}\ m \Longrightarrow (a,b) \in \textit{Rel}\ \hat{}\hat{}\ n$

$\langle proof \rangle$

lemma *SN-on-induct-acc-style* [*consumes 1*, *case-names IH*]:
assumes *sn*: *SN-on R* $\{a\}$
and *IH*: $\bigwedge x$. *SN-on R* $\{x\} \Longrightarrow [\![\bigwedge y.\ (x, y) \in R \Longrightarrow P\ y]\!] \Longrightarrow P\ x$
shows $P\ a$
$\langle proof \rangle$

lemma *partially-localize-CR*:
$CR\ r \longleftrightarrow (\forall\ x\ y\ z.\ (x, y) \in r \wedge (x, z) \in r^* \longrightarrow (y, z) \in join\ r)$
$\langle proof \rangle$

definition *strongly-confluent-on* :: $'a\ rel \Rightarrow 'a\ set \Rightarrow bool$
where
strongly-confluent-on r A $\longleftrightarrow$
$(\forall x \in A.\ \forall y\ z.\ (x, y) \in r \wedge (x, z) \in r \longrightarrow (\exists u.\ (y, u) \in r^* \wedge (z, u) \in r^=))$

abbreviation *strongly-confluent* :: $'a\ rel \Rightarrow bool$
where
strongly-confluent r $\equiv$ *strongly-confluent-on r UNIV*

lemma *strongly-confluent-on-E11*:
strongly-confluent-on r A $\Longrightarrow x \in A \Longrightarrow (x, y) \in r \Longrightarrow (x, z) \in r \Longrightarrow$
$\exists u.\ (y, u) \in r^* \wedge (z, u) \in r^=$
$\langle proof \rangle$

lemma *strongly-confluentI* [*intro*]:
$[\![\bigwedge x\ y\ z.\ (x, y) \in r \Longrightarrow (x, z) \in r \Longrightarrow \exists u.\ (y, u) \in r^* \wedge (z, u) \in r^=]\!] \Longrightarrow$ *strongly-confluent r*
$\langle proof \rangle$

lemma *strongly-confluent-E1n*:
assumes *scr*: *strongly-confluent r*
shows $(x, y) \in r^= \Longrightarrow (x, z) \in r$ ^^ $n \Longrightarrow \exists u.\ (y, u) \in r^* \wedge (z, u) \in r^=$
$\langle proof \rangle$

lemma *strong-confluence-imp-CR*:
assumes *strongly-confluent r*
shows *CR r*
$\langle proof \rangle$

lemma *WCR-alt-def*: $WCR\ A \longleftrightarrow A^{-1}\ O\ A \subseteq A^{\downarrow}$ $\langle proof \rangle$

lemma *NF-imp-SN-on*: $a \in NF\ R \Longrightarrow$ *SN-on R* $\{a\}$ $\langle proof \rangle$

lemma *Union-sym*: $(s, t) \in (\bigcup i \le n.\ (S\ i)^{\leftrightarrow}) \longleftrightarrow (t, s) \in (\bigcup i \le n.\ (S\ i)^{\leftrightarrow})$ $\langle proof \rangle$

lemma *peak-iff*: $(x, y) \in A^{-1}\ O\ B \longleftrightarrow (\exists u.\ (u, x) \in A \wedge (u, y) \in B)$ ⟨*proof*⟩

lemma *CR-NF-conv*:
assumes $CR\ r$ **and** $t \in NF\ r$ **and** $(u, t) \in r^{\leftrightarrow *}$
shows $(u, t) \in r^{!}$
⟨*proof*⟩

lemma *NF-join-imp-reach*:
assumes $y \in NF\ A$ **and** $(x, y) \in A^{\downarrow}$
shows $(x, y) \in A^{*}$
⟨*proof*⟩

lemma *conversion-O-conversion* [*simp*]:
$A^{\leftrightarrow *}\ O\ A^{\leftrightarrow *} = A^{\leftrightarrow *}$
⟨*proof*⟩

lemma *trans-O-iff*: $trans\ A \longleftrightarrow A\ O\ A \subseteq A$ ⟨*proof*⟩
lemma *refl-O-iff*: $refl\ A \longleftrightarrow Id \subseteq A$ ⟨*proof*⟩

lemma *relpow-Suc*: $r \mathbin{\hat{}\hat{}} Suc\ n = r\ O\ r \mathbin{\hat{}\hat{}} n$
⟨*proof*⟩

lemma *converse-power*: **fixes** $r :: 'a\ rel$ **shows** $(r^{-1})\hat{}\hat{}n = (r\hat{}\hat{}n)^{-1}$
⟨*proof*⟩

lemma *conversion-mono*: $A \subseteq B \Longrightarrow A^{\leftrightarrow *} \subseteq B^{\leftrightarrow *}$
⟨*proof*⟩

lemma *conversion-conversion-idemp* [*simp*]: $(A^{\leftrightarrow *})^{\leftrightarrow *} = A^{\leftrightarrow *}$
⟨*proof*⟩

lemma *lower-set-imp-not-SN-on*:
assumes $s \in X\ \forall t \in X.\ \exists u \in X.\ (t,u) \in R$ **shows** $\neg\ SN\text{-}on\ R\ \{s\}$
⟨*proof*⟩

lemma *SN-on-Image-rtrancl-iff*[*simp*]: $SN\text{-}on\ R\ (R^{*}\ ``\ X) \longleftrightarrow SN\text{-}on\ R\ X$ (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *O-mono1*: $R \subseteq R' \Longrightarrow S\ O\ R \subseteq S\ O\ R'$ ⟨*proof*⟩
lemma *O-mono2*: $R \subseteq R' \Longrightarrow R\ O\ T \subseteq R'\ O\ T$ ⟨*proof*⟩

lemma *rtrancl-O-shift*: $(S\ O\ R)^{*}\ O\ S = S\ O\ (R\ O\ S)^{*}$

⟨*proof*⟩

lemma *O-rtrancl-O-O*: $R\ O\ (S\ O\ R)^{*}\ O\ S = (R\ O\ S)^{+}$
⟨*proof*⟩

lemma *SN-on-subset-SN-terms*:
assumes *SN*: *SN-on R X* **shows** $X \subseteq \{x.\ SN\text{-}on\ R\ \{x\}\}$
⟨*proof*⟩

lemma *SN-on-Un2*:
assumes *SN-on R X* **and** *SN-on R Y* **shows** *SN-on R* $(X \cup Y)$
⟨*proof*⟩

lemma *SN-on-UN*:
assumes $\bigwedge x.$ *SN-on R* $(X\ x)$ **shows** *SN-on R* $(\bigcup x.\ X\ x)$
⟨*proof*⟩

lemma *Image-subsetI*: $R \subseteq R' \Longrightarrow R\ ``\ X \subseteq R'\ ``\ X$ ⟨*proof*⟩

lemma *SN-on-O-comm*:
assumes *SN*: *SN-on* $((R :: ('a \times 'b)\ set)\ O\ (S :: ('b \times 'a)\ set))\ (S\ ``\ X)$
shows *SN-on* $(S\ O\ R)\ X$
⟨*proof*⟩

lemma *SN-O-comm*: $SN\ (R\ O\ S) \longleftrightarrow SN\ (S\ O\ R)$
⟨*proof*⟩

lemma *chain-mono*: **assumes** $R' \subseteq R$ *chain* R' *seq* **shows** *chain R seq*
⟨*proof*⟩

context
fixes *S R*
assumes *push*: $S\ O\ R \subseteq R\ O\ S^*$
begin

lemma *rtrancl-O-push*: $S^*\ O\ R \subseteq R\ O\ S^*$
⟨*proof*⟩

lemma *rtrancl-U-push*: $(S \cup R)^* = R^*\ O\ S^*$
⟨*proof*⟩

lemma *SN-on-O-push*:
assumes *SN*: *SN-on R X* **shows** *SN-on* $(R\ O\ S^*)\ X$
⟨*proof*⟩

lemma *SN-on-Image-push*:
assumes *SN*: *SN-on R X* **shows** *SN-on R* $(S^*\ ``\ X)$
⟨*proof*⟩

end

lemma *not-SN-onI*[*intro*]: $f\ 0 \in X \Longrightarrow chain\ R\ f \Longrightarrow \neg\ SN\text{-}on\ R\ X$
⟨*proof*⟩

lemma *shift-comp*[*simp*]: *shift* ($f \circ seq$) $n = f \circ$ (*shift seq n*) ⟨*proof*⟩

lemma *Id-on-union*: *Id-on* $(A \cup B)$ = *Id-on* $A \cup$ *Id-on* B ⟨*proof*⟩

lemma *relpow-union-cases*: $(a,d) \in (A \cup B)$^^$n \Longrightarrow (a,d) \in B$^^$n \vee (\exists\ b\ c\ k\ m.$
$(a,b) \in B$^^$k \wedge (b,c) \in A \wedge (c,d) \in (A \cup B)$^^$m \wedge n = Suc\ (k + m))$
⟨*proof*⟩

lemma *trans-refl-imp-rtrancl-id*:
  assumes *trans r refl r*
  shows $r^* = r$
⟨*proof*⟩

lemma *trans-refl-imp-O-id*:
  assumes *trans r refl r*
  shows $r\ O\ r = r$
⟨*proof*⟩

lemma *relcomp3-I*:
  assumes $(t, u) \in A$ **and** $(s, t) \in B$ **and** $(u, v) \in B$
  shows $(s, v) \in B\ O\ A\ O\ B$
  ⟨*proof*⟩

lemma *relcomp3-transI*:
  assumes *trans B* **and** $(t, u) \in B\ O\ A\ O\ B$ **and** $(s, t) \in B$ **and** $(u, v) \in B$
  shows $(s, v) \in B\ O\ A\ O\ B$
⟨*proof*⟩

lemmas *converse-inward* = *rtrancl-converse*[*symmetric*] *converse-Un converse-UNION converse-relcomp*
  converse-converse converse-Id

lemma *qc-SN-relto-iff*:
  assumes $r\ O\ s \subseteq s\ O\ (s \cup r)^*$
  shows $SN\ (r^*\ O\ s\ O\ r^*) = SN\ s$
⟨*proof*⟩

lemma *conversion-empty* [*simp*]: *conversion* {} = *Id*
  ⟨*proof*⟩

lemma *symcl-idemp* [*simp*]: $(r^{\leftrightarrow})^{\leftrightarrow} = r^{\leftrightarrow}$ ⟨*proof*⟩

end

3 Relative Rewriting

theory *Relative-Rewriting*
imports *Abstract-Rewriting*
begin

Considering a relation R relative to another relation S, i.e., R-steps may be preceded and followed by arbitrary many S-steps.

abbreviation (*input*) *relto* :: *'a rel* ⇒ *'a rel* ⇒ *'a rel* **where**
relto R S ≡ *S^** *O R O S^**

definition *SN-rel-on* :: *'a rel* ⇒ *'a rel* ⇒ *'a set* ⇒ *bool* **where**
SN-rel-on R S ≡ *SN-on* (*relto R S*)

definition *SN-rel-on-alt* :: *'a rel* ⇒ *'a rel* ⇒ *'a set* ⇒ *bool* **where**
SN-rel-on-alt R S T = (∀ *f*. *chain* (*R* ∪ *S*) *f* ∧ *f 0* ∈ *T* ⟶ ¬ (*INFM j*. (*f j*, *f* (*Suc j*)) ∈ *R*))

abbreviation *SN-rel* :: *'a rel* ⇒ *'a rel* ⇒ *bool* **where**
SN-rel R S ≡ *SN-rel-on R S UNIV*

abbreviation *SN-rel-alt* :: *'a rel* ⇒ *'a rel* ⇒ *bool* **where**
SN-rel-alt R S ≡ *SN-rel-on-alt R S UNIV*

lemma *relto-absorb* [*simp*]: *relto R E O* E^* = *relto R E* E^* *O relto R E* = *relto R E*
⟨*proof*⟩

lemma *steps-preserve-SN-on-relto*:
assumes *steps*: (*a*, *b*) ∈ (*R* ∪ *S*)^*
and *SN*: *SN-on* (*relto R S*) {*a*}
shows *SN-on* (*relto R S*) {*b*}
⟨*proof*⟩

lemma *step-preserves-SN-on-relto*: **assumes** *st*: (*s*,*t*) ∈ *R* ∪ *E*
and *SN*: *SN-on* (*relto R E*) {*s*}
shows *SN-on* (*relto R E*) {*t*}
⟨*proof*⟩

lemma *SN-rel-on-imp-SN-rel-on-alt*: *SN-rel-on R S T* ⟹ *SN-rel-on-alt R S T*
⟨*proof*⟩

lemma *SN-rel-on-alt-imp-SN-rel-on*: *SN-rel-on-alt R S T* ⟹ *SN-rel-on R S T*
⟨*proof*⟩

lemma *SN-rel-on-conv*: *SN-rel-on* = *SN-rel-on-alt*
⟨*proof*⟩

lemmas *SN-rel-defs* = *SN-rel-on-def SN-rel-on-alt-def*

lemma *SN-rel-on-alt-r-empty* : *SN-rel-on-alt* {} *S T*
⟨*proof*⟩

lemma *SN-rel-on-alt-s-empty* : *SN-rel-on-alt R* {} = *SN-on R*

⟨*proof*⟩

lemma *SN-rel-on-mono'*:
assumes *R*: $R \subseteq R'$ **and** *S*: $S \subseteq R' \cup S'$ **and** *SN*: *SN-rel-on* R' S' *T*
shows *SN-rel-on R S T*
⟨*proof*⟩

lemma *relto-mono*:
assumes $R \subseteq R'$ **and** $S \subseteq S'$
shows *relto R S* $\subseteq$ *relto* R' S'
⟨*proof*⟩

lemma *SN-rel-on-mono*:
assumes *R*: $R \subseteq R'$ **and** *S*: $S \subseteq S'$
and *SN*: *SN-rel-on* R' S' *T*
shows *SN-rel-on R S T*
⟨*proof*⟩

lemmas *SN-rel-on-alt-mono* = *SN-rel-on-mono*[*unfolded SN-rel-on-conv*]

lemma *SN-rel-on-imp-SN-on*:
assumes *SN-rel-on R S T* **shows** *SN-on R T*
⟨*proof*⟩

lemma *relto-Id*: *relto R* ($S \cup$ *Id*) = *relto R S* ⟨*proof*⟩

lemma *SN-rel-on-Id*:
shows *SN-rel-on R* ($S \cup$ *Id*) *T* = *SN-rel-on R S T*
⟨*proof*⟩

lemma *SN-rel-on-empty*[*simp*]: *SN-rel-on R* {} *T* = *SN-on R T*
⟨*proof*⟩

lemma *SN-rel-on-ideriv*: *SN-rel-on R S T* = ($\neg$ ($\exists$ *as. ideriv R S as* $\wedge$ *as 0* $\in$ *T*))
(**is** *?L* = *?R*)
⟨*proof*⟩

lemma *SN-rel-to-SN-rel-alt*: *SN-rel R S* $\Longrightarrow$ *SN-rel-alt R S*
⟨*proof*⟩

lemma *SN-rel-alt-to-SN-rel* : *SN-rel-alt R S* $\Longrightarrow$ *SN-rel R S*
⟨*proof*⟩

lemma *SN-rel-alt-r-empty* : *SN-rel-alt* {} *S*
⟨*proof*⟩

lemma *SN-rel-alt-s-empty* : *SN-rel-alt R* {} = *SN R*
⟨*proof*⟩

lemma *SN-rel-mono′*:
$R \subseteq R' \Longrightarrow S \subseteq R' \cup S' \Longrightarrow$ *SN-rel* R' S' $\Longrightarrow$ *SN-rel* R S
⟨*proof*⟩

lemma *SN-rel-mono*:
assumes R: $R \subseteq R'$ **and** S: $S \subseteq S'$ **and** *SN*: *SN-rel* R' S'
shows *SN-rel R S*
⟨*proof*⟩

lemmas *SN-rel-alt-mono* = *SN-rel-mono*[*unfolded SN-rel-on-conv*]

lemma *SN-rel-imp-SN* : **assumes** *SN-rel R S* **shows** *SN R*
⟨*proof*⟩

lemma *relto-trancl-conv* : (*relto R S*)^+ = ((*R* ∪ *S*))^* *O R O* ((*R* ∪ *S*))^*
⟨*proof*⟩

lemma *SN-rel-Id*:
shows *SN-rel R* (*S* ∪ *Id*) = *SN-rel R S*
⟨*proof*⟩

lemma *relto-rtrancl*: *relto R* (*S*^*) = *relto R S* ⟨*proof*⟩

lemma *SN-rel-empty*[*simp*]: *SN-rel R* {} = *SN R*
⟨*proof*⟩

lemma *SN-rel-ideriv*: *SN-rel R S* = (¬ (∃ *as. ideriv R S as*)) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *SN-rel-map*:
fixes R Rw R' Rw' :: ′*a rel*
defines A: $A \equiv R' \cup Rw'$
assumes *SN*: *SN-rel* R' Rw'
and R: $\bigwedge s\ t.\ (s,t) \in R \Longrightarrow (f\ s,\ f\ t) \in A$^* O R' O A^*
and Rw: $\bigwedge s\ t.\ (s,t) \in Rw \Longrightarrow (f\ s,\ f\ t) \in A$^*
shows *SN-rel R Rw*
⟨*proof*⟩

datatype *SN-rel-ext-type* = *top-s* | *top-ns* | *normal-s* | *normal-ns*

fun *SN-rel-ext-step* :: ′*a rel* ⇒ ′*a rel* ⇒ ′*a rel* ⇒ ′*a rel* ⇒ *SN-rel-ext-type* ⇒ ′*a rel*
where
SN-rel-ext-step P Pw R Rw top-s = *P*
| *SN-rel-ext-step P Pw R Rw top-ns* = *Pw*
| *SN-rel-ext-step P Pw R Rw normal-s* = *R*
| *SN-rel-ext-step P Pw R Rw normal-ns* = *Rw*

definition *SN-rel-ext* :: ′*a rel* ⇒ ′*a rel* ⇒ ′*a rel* ⇒ ′*a rel* ⇒ (′*a* ⇒ *bool*) ⇒ *bool*

where
$SN\text{-}rel\text{-}ext\ P\ Pw\ R\ Rw\ M \equiv (\neg\ (\exists f\ t.$
$(\forall\ i.\ (f\ i,\ f\ (Suc\ i)) \in SN\text{-}rel\text{-}ext\text{-}step\ P\ Pw\ R\ Rw\ (t\ i))$
$\wedge\ (\forall\ i.\ M\ (f\ i))$
$\wedge\ (INFM\ i.\ t\ i \in \{top\text{-}s,top\text{-}ns\})$
$\wedge\ (INFM\ i.\ t\ i \in \{top\text{-}s,normal\text{-}s\})))$

lemma *SN-rel-ext-step-mono*: **assumes** $P \subseteq P'$ $Pw \subseteq Pw'$ $R \subseteq R'$ $Rw \subseteq Rw'$
shows $SN\text{-}rel\text{-}ext\text{-}step\ P\ Pw\ R\ Rw\ t \subseteq SN\text{-}rel\text{-}ext\text{-}step\ P'\ Pw'\ R'\ Rw'\ t$
⟨*proof*⟩

lemma *SN-rel-ext-mono*: **assumes** *subset*: $P \subseteq P'$ $Pw \subseteq Pw'$ $R \subseteq R'$ $Rw \subseteq Rw'$ **and**
SN: $SN\text{-}rel\text{-}ext\ P'\ Pw'\ R'\ Rw'\ M$ **shows** $SN\text{-}rel\text{-}ext\ P\ Pw\ R\ Rw\ M$
⟨*proof*⟩

lemma *SN-rel-ext-trans*:
fixes $P\ Pw\ R\ Rw :: 'a\ rel$ **and** $M :: 'a \Rightarrow bool$
defines M': $M' \equiv \{(s,t).\ M\ t\}$
defines A: $A \equiv (P \cup Pw \cup R \cup Rw) \cap M'$
assumes $SN\text{-}rel\text{-}ext\ P\ Pw\ R\ Rw\ M$
shows $SN\text{-}rel\text{-}ext\ (A\hat{~}*\ O\ (P \cap M')\ O\ A\hat{~}*)\ (A\hat{~}*\ O\ ((P \cup Pw) \cap M')\ O\ A\hat{~}*)\ (A\hat{~}*\ O\ ((P \cup R) \cap M')\ O\ A\hat{~}*)\ (A\hat{~}*)\ M$ (**is** $SN\text{-}rel\text{-}ext\ ?P\ ?Pw\ ?R\ ?Rw\ M$)
⟨*proof*⟩

lemma *SN-rel-ext-map*: **fixes** $P\ Pw\ R\ Rw\ P'\ Pw'\ R'\ Rw' :: 'a\ rel$ **and** $M\ M' :: 'a \Rightarrow bool$
defines Ms: $Ms \equiv \{(s,t).\ M'\ t\}$
defines A: $A \equiv (P' \cup Pw' \cup R' \cup Rw') \cap Ms$
assumes *SN*: $SN\text{-}rel\text{-}ext\ P'\ Pw'\ R'\ Rw'\ M'$
and P: $\bigwedge\ s\ t.\ M\ s \Longrightarrow M\ t \Longrightarrow (s,t) \in P \Longrightarrow (f\ s,\ f\ t) \in (A\hat{~}*\ O\ (P' \cap Ms)\ O\ A\hat{~}*) \wedge I\ t$
and Pw: $\bigwedge\ s\ t.\ M\ s \Longrightarrow M\ t \Longrightarrow (s,t) \in Pw \Longrightarrow (f\ s,\ f\ t) \in (A\hat{~}*\ O\ ((P' \cup Pw') \cap Ms)\ O\ A\hat{~}*) \wedge I\ t$
and R: $\bigwedge\ s\ t.\ I\ s \Longrightarrow M\ s \Longrightarrow M\ t \Longrightarrow (s,t) \in R \Longrightarrow (f\ s,\ f\ t) \in (A\hat{~}*\ O\ ((P' \cup R') \cap Ms)\ O\ A\hat{~}*) \wedge I\ t$
and Rw: $\bigwedge\ s\ t.\ I\ s \Longrightarrow M\ s \Longrightarrow M\ t \Longrightarrow (s,t) \in Rw \Longrightarrow (f\ s,\ f\ t) \in A\hat{~}* \wedge I\ t$
shows $SN\text{-}rel\text{-}ext\ P\ Pw\ R\ Rw\ M$
⟨*proof*⟩

lemma *SN-rel-ext-map-min*: **fixes** $P\ Pw\ R\ Rw\ P'\ Pw'\ R'\ Rw' :: 'a\ rel$ **and** $M\ M' :: 'a \Rightarrow bool$
defines Ms: $Ms \equiv \{(s,t).\ M'\ t\}$
defines A: $A \equiv P' \cap Ms \cup Pw' \cap Ms \cup R' \cup Rw'$
assumes *SN*: $SN\text{-}rel\text{-}ext\ P'\ Pw'\ R'\ Rw'\ M'$
and M: $\bigwedge\ t.\ M\ t \Longrightarrow M'\ (f\ t)$
and M': $\bigwedge\ s\ t.\ M'\ s \Longrightarrow (s,t) \in R' \cup Rw' \Longrightarrow M'\ t$

and *P*: $\bigwedge$ *s t*. *M s* $\Longrightarrow$ *M t* $\Longrightarrow$ *M′* (*f s*) $\Longrightarrow$ *M′* (*f t*) $\Longrightarrow$ (*s*,*t*) $\in$ *P* $\Longrightarrow$ (*f s*, *f t*) $\in$ (*A*^* *O* (*P′* $\cap$ *Ms*) *O A*^*) $\wedge$ *I t*
and *Pw*: $\bigwedge$ *s t*. *M s* $\Longrightarrow$ *M t* $\Longrightarrow$ *M′* (*f s*) $\Longrightarrow$ *M′* (*f t*) $\Longrightarrow$ (*s*,*t*) $\in$ *Pw* $\Longrightarrow$ (*f s*, *f t*) $\in$ (*A*^* *O* (*P′* $\cap$ *Ms* $\cup$ *Pw′* $\cap$ *Ms*) *O A*^*) $\wedge$ *I t*
and *R*: $\bigwedge$ *s t*. *I s* $\Longrightarrow$ *M s* $\Longrightarrow$ *M t* $\Longrightarrow$ *M′* (*f s*) $\Longrightarrow$ *M′* (*f t*) $\Longrightarrow$ (*s*,*t*) $\in$ *R* $\Longrightarrow$ (*f s*, *f t*) $\in$ (*A*^* *O* (*P′* $\cap$ *Ms* $\cup$ *R′*) *O A*^*) $\wedge$ *I t*
and *Rw*: $\bigwedge$ *s t*. *I s* $\Longrightarrow$ *M s* $\Longrightarrow$ *M t* $\Longrightarrow$ *M′* (*f s*) $\Longrightarrow$ *M′* (*f t*) $\Longrightarrow$ (*s*,*t*) $\in$ *Rw* $\Longrightarrow$ (*f s*, *f t*) $\in$ *A*^* $\wedge$ *I t*
shows *SN-rel-ext P Pw R Rw M*
⟨*proof*⟩

lemma *SN-relto-imp-SN-rel*: *SN* (*relto R S*) $\Longrightarrow$ *SN-rel R S*
⟨*proof*⟩

lemma *rtrancl-list-conv*:
((*s*,*t*) $\in$ *R*^*) =
($\exists$ *list*. *last* (*s* # *list*) = *t* $\wedge$ ($\forall$ *i*. *i* < *length list* $\longrightarrow$ ((*s* # *list*) ! *i*, (*s* # *list*) ! *Suc i*) $\in$ *R*)) (**is** *?l* = *?r*)
⟨*proof*⟩

fun *choice* :: (*nat* ⇒ *′a list*) ⇒ *nat* ⇒ (*nat* × *nat*) **where**
choice f 0 = (*0*,*0*)
| *choice f* (*Suc n*) = (*let* (*i*, *j*) = *choice f n in*
if Suc j < *length* (*f i*)
then (*i*, *Suc j*)
else (*Suc i*, *0*))

lemma *SN-rel-imp-SN-relto* : *SN-rel R S* $\Longrightarrow$ *SN* (*relto R S*)
⟨*proof*⟩

hide-const *choice*

lemma *SN-relto-SN-rel-conv*: *SN* (*relto R S*) = *SN-rel R S*
⟨*proof*⟩

lemma *SN-rel-empty1*: *SN-rel* {} *S*
⟨*proof*⟩

lemma *SN-rel-empty2*: *SN-rel R* {} = *SN R*
⟨*proof*⟩

lemma *SN-relto-mono*:
assumes *R*: *R* ⊆ *R′* **and** *S*: *S* ⊆ *S′*
and *SN*: *SN* (*relto R′ S′*)
shows *SN* (*relto R S*)
⟨*proof*⟩

lemma *SN-relto-imp-SN*:
assumes *SN* (*relto R S*) **shows** *SN R*
⟨*proof*⟩

lemma *SN-relto-Id*:
SN (*relto R* (*S* ∪ *Id*)) = *SN* (*relto R S*)
⟨*proof*⟩

Termination inheritance by transitivity (see, e.g., Geser's thesis).

lemma *trans-subset-SN*:
assumes *trans R* **and** *R* ⊆ (*r* ∪ *s*) **and** *SN r* **and** *SN s*
shows *SN R*
⟨*proof*⟩

lemma *SN-Un-conv*:
assumes *trans* (*r* ∪ *s*)
shows *SN* (*r* ∪ *s*) ⟷ *SN r* ∧ *SN s*
(**is** *SN ?r* ⟷ *?rhs*)
⟨*proof*⟩

lemma *SN-relto-Un*:
SN (*relto* (*R* ∪ *S*) *Q*) ⟷ *SN* (*relto R* (*S* ∪ *Q*)) ∧ *SN* (*relto S Q*)
(**is** *SN ?a* ⟷ *SN ?b* ∧ *SN ?c*)
⟨*proof*⟩

lemma *SN-relto-split*:
assumes *SN* (*relto r* (*s* ∪ *q2*) ∪ *relto q1* (*s* ∪ *q2*)) (**is** *SN ?a*)
and *SN* (*relto s q2*) (**is** *SN ?b*)
shows *SN* (*relto r* (*q1* ∪ *q2*) ∪ *relto s* (*q1* ∪ *q2*)) (**is** *SN ?c*)
⟨*proof*⟩

lemma *relto-trancl-subset*: **assumes** *a* ⊆ *c* **and** *b* ⊆ *c* **shows** *relto a b* ⊆ *c*^+
⟨*proof*⟩

An explicit version of *relto* which mentions all intermediate terms

inductive *relto-fun* :: *'a rel* ⇒ *'a rel* ⇒ *nat* ⇒ (*nat* ⇒ *'a*) ⇒ (*nat* ⇒ *bool*) ⇒ *nat* ⇒ *'a* × *'a* ⇒ *bool* **where**
relto-fun: *as 0* = *a* ⟹ *as m* = *b* ⟹
(⋀ *i*. *i* < *m* ⟹
(*sel i* ⟶ (*as i*, *as* (*Suc i*)) ∈ *A*) ∧ (¬ *sel i* ⟶ (*as i*, *as* (*Suc i*)) ∈ *B*))
⟹ *n* = *card* { *i* . *i* < *m* ∧ *sel i*}
⟹ (*n* = *0* ⟷ *m* = *0*) ⟹ *relto-fun A B n as sel m* (*a*,*b*)

lemma *relto-funD*: **assumes** *relto-fun A B n as sel m* (*a*,*b*)
shows *as 0* = *a as m* = *b*
⋀ *i*. *i* < *m* ⟹ *sel i* ⟹ (*as i*, *as* (*Suc i*)) ∈ *A*
⋀ *i*. *i* < *m* ⟹ ¬ *sel i* ⟹ (*as i*, *as* (*Suc i*)) ∈ *B*
n = *card* { *i* . *i* < *m* ∧ *sel i*}
n = *0* ⟷ *m* = *0*

⟨*proof*⟩

lemma *relto-fun-refl*: $\exists$ *as sel. relto-fun A B 0 as sel 0* (*a*,*a*)
⟨*proof*⟩

lemma *relto-into-relto-fun*: **assumes** (*a*,*b*) $\in$ *relto A B*
shows $\exists$ *as sel m. relto-fun A B* (*Suc 0*) *as sel m* (*a*,*b*)
⟨*proof*⟩

lemma *relto-fun-trans*: **assumes** *ab*: *relto-fun A B n1 as1 sel1 m1* (*a*,*b*)
and *bc*: *relto-fun A B n2 as2 sel2 m2* (*b*,*c*)
shows $\exists$ *as sel. relto-fun A B* (*n1* + *n2*) *as sel* (*m1* + *m2*) (*a*,*c*)
⟨*proof*⟩

lemma *reltos-into-relto-fun*: **assumes** (*a*,*b*) $\in$ (*relto A B*)^^*n*
shows $\exists$ *as sel m. relto-fun A B n as sel m* (*a*,*b*)
⟨*proof*⟩

lemma *relto-fun-into-reltos*: **assumes** *relto-fun A B n as sel m* (*a*,*b*)
shows (*a*,*b*) $\in$ (*relto A B*)^^*n*
⟨*proof*⟩

lemma *relto-relto-fun-conv*: ((*a*,*b*) $\in$ (*relto A B*)^^*n*) = ($\exists$ *as sel m. relto-fun A B n as sel m* (*a*,*b*))
⟨*proof*⟩

lemma *relto-fun-intermediate*: **assumes** $A \subseteq C$ **and** $B \subseteq C$
and *rf*: *relto-fun A B n as sel m* (*a*,*b*)
shows $i \leq m \Longrightarrow$ (*a*,*as i*) $\in$ *C*^*
⟨*proof*⟩

lemma *not-SN-on-rel-succ*:
assumes $\neg$ *SN-on* (*relto R E*) $\{s\}$
shows $\exists t\ u.\ (s,\ t) \in E^* \wedge (t,\ u) \in R \wedge \neg$ *SN-on* (*relto R E*) $\{u\}$
⟨*proof*⟩

lemma *SN-on-relto-relcomp*: *SN-on* (*relto R S*) *T* = *SN-on* (S^* *O R*) *T* (**is** *?L T* = *?R T*)
⟨*proof*⟩

lemma *trans-relto*:
assumes *trans*: *trans R* **and** *S O R* $\subseteq$ *R O S*
shows *trans* (*relto R S*)
⟨*proof*⟩

lemma *relative-ending*:
assumes *chain*: *chain* ($R \cup S$) *t*
and *t0*: *t 0* $\in$ *X*
and *SN*: *SN-on* (*relto R S*) *X*

shows $\exists j. \forall i \geq j. (t\ i,\ t\ (Suc\ i)) \in S - R$
$\langle proof \rangle$

from Geser's thesis [p.32, Corollary-1], generalized for *SN-on*.

lemma *SN-on-relto-Un*:
assumes *closure*: $relto\ (R \cup R')\ S\ `` X \subseteq X$
shows $SN\text{-}on\ (relto\ (R \cup R')\ S)\ X \longleftrightarrow SN\text{-}on\ (relto\ R\ (R' \cup S))\ X \wedge SN\text{-}on\ (relto\ R'\ S)\ X$
(**is** $?c \longleftrightarrow ?a \wedge ?b$)
$\langle proof \rangle$

lemma *SN-on-Un*: $(R \cup R')``X \subseteq X \Longrightarrow SN\text{-}on\ (R \cup R')\ X \longleftrightarrow SN\text{-}on\ (relto\ R\ R')\ X \wedge SN\text{-}on\ R'\ X$
$\langle proof \rangle$

end

4 Strongly Normalizing Orders

theory *SN-Orders*
imports *Abstract-Rewriting*
begin

We define several classes of orders which are used to build ordered semirings. Note that we do not use Isabelle's preorders since the condition $x > y = x \geq y \wedge y \not\geq x$ is sometimes not applicable. E.g., for δ-orders over the rationals we have $0.2 \geq 0.1 \wedge 0.1 \not\geq 0.2$, but $0.2 >_\delta 0.1$ does not hold if δ is larger than 0.1.

class *non-strict-order* = *ord* +
assumes *ge-refl*: $x \geq (x :: 'a)$
and *ge-trans*[*trans*]: $[\![x \geq y; (y :: 'a) \geq z]\!] \Longrightarrow x \geq z$
and *max-comm*: $max\ x\ y = max\ y\ x$
and *max-ge-x*[*intro*]: $max\ x\ y \geq x$
and *max-id*: $x \geq y \Longrightarrow max\ x\ y = x$
and *max-mono*: $x \geq y \Longrightarrow max\ z\ x \geq max\ z\ y$
begin
lemma *max-ge-y*[*intro*]: $max\ x\ y \geq y$
$\langle proof \rangle$

lemma *max-mono2*: $x \geq y \Longrightarrow max\ x\ z \geq max\ y\ z$
$\langle proof \rangle$
end

class *ordered-ab-semigroup* = *non-strict-order* + *ab-semigroup-add* + *monoid-add* +
assumes *plus-left-mono*: $x \geq y \Longrightarrow x + z \geq y + z$

lemma *plus-right-mono*: $y \geq (z :: 'a :: ordered\text{-}ab\text{-}semigroup) \Longrightarrow x + y \geq x + z$

⟨*proof*⟩

class *ordered-semiring-0* = *ordered-ab-semigroup* + *semiring-0* +
assumes *times-left-mono*: $z \geq 0 \Longrightarrow x \geq y \Longrightarrow x * z \geq y * z$
and *times-right-mono*: $x \geq 0 \Longrightarrow y \geq z \Longrightarrow x * y \geq x * z$
and *times-left-anti-mono*: $x \geq y \Longrightarrow 0 \geq z \Longrightarrow y * z \geq x * z$

class *ordered-semiring-1* = *ordered-semiring-0* + *semiring-1* +
assumes *one-ge-zero*: $1 \geq 0$

We do not use a class to define order-pairs of a strict and a weak-order since often we have parametric strict orders, e.g. on rational numbers there are several orders $>$ where $x > y = x \geq y + \delta$ for some parameter δ

locale *order-pair* =
fixes *gt* :: *'a* :: {*non-strict-order*,*zero*} ⇒ *'a* ⇒ *bool* (**infix** ≻ *50*)
and *default* :: *'a*
assumes *compat*[*trans*]: $[\![x \geq y; y \succ z]\!] \Longrightarrow x \succ z$
and *compat2*[*trans*]: $[\![x \succ y; y \geq z]\!] \Longrightarrow x \succ z$
and *gt-imp-ge*: $x \succ y \Longrightarrow x \geq y$
and *default-ge-zero*: *default* ≥ *0*
begin
lemma *gt-trans*[*trans*]: $[\![x \succ y; y \succ z]\!] \Longrightarrow x \succ z$
⟨*proof*⟩
end

locale *one-mono-ordered-semiring-1* = *order-pair gt*
for *gt* :: *'a* :: *ordered-semiring-1* ⇒ *'a* ⇒ *bool* (**infix** ≻ *50*) +
assumes *plus-gt-left-mono*: $x \succ y \Longrightarrow x + z \succ y + z$
and *default-gt-zero*: *default* ≻ *0*
begin
lemma *plus-gt-right-mono*: $x \succ y \Longrightarrow a + x \succ a + y$
⟨*proof*⟩

lemma *plus-gt-both-mono*: $x \succ y \Longrightarrow a \succ b \Longrightarrow x + a \succ y + b$
⟨*proof*⟩
end

locale *SN-one-mono-ordered-semiring-1* = *one-mono-ordered-semiring-1* + *order-pair* +
assumes *SN*: *SN* {(*x*,*y*) . $y \geq 0 \wedge x \succ y$}

locale *SN-strict-mono-ordered-semiring-1* = *SN-one-mono-ordered-semiring-1* +
fixes *mono* :: *'a* :: *ordered-semiring-1* ⇒ *bool*
assumes *mono*: $[\![mono\ x; y \succ z; x \geq 0]\!] \Longrightarrow x * y \succ x * z$

locale *both-mono-ordered-semiring-1* = *order-pair gt*
for *gt* :: *'a* :: *ordered-semiring-1* ⇒ *'a* ⇒ *bool* (**infix** ≻ *50*) +
fixes *arc-pos* :: *'a* ⇒ *bool*

assumes *plus-gt-both-mono*: ⟦*x* ≻ *y*; *z* ≻ *u*⟧ ⟹ *x* + *z* ≻ *y* + *u*
and *times-gt-left-mono*: *x* ≻ *y* ⟹ *x* ∗ *z* ≻ *y* ∗ *z*
and *times-gt-right-mono*: *y* ≻ *z* ⟹ *x* ∗ *y* ≻ *x* ∗ *z*
and *zero-leastI*: *x* ≻ *0*
and *zero-leastII*: *0* ≻ *x* ⟹ *x* = *0*
and *zero-leastIII*: (*x* :: *'a*) ≥ *0*
and *arc-pos-one*: *arc-pos* (*1* :: *'a*)
and *arc-pos-default*: *arc-pos default*
and *arc-pos-zero*: ¬ *arc-pos 0*
and *arc-pos-plus*: *arc-pos x* ⟹ *arc-pos* (*x* + *y*)
and *arc-pos-mult*: ⟦*arc-pos x*; *arc-pos y*⟧ ⟹ *arc-pos* (*x* ∗ *y*)
and *not-all-ge*: ⋀ *c d*. *arc-pos d* ⟹ ∃ *e*. *e* ≥ *0* ∧ *arc-pos e* ∧ ¬ (*c* ≥ *d* ∗ *e*)
begin
lemma *max0-id*: *max 0* (*x* :: *'a*) = *x*
⟨*proof*⟩
end

locale *SN-both-mono-ordered-semiring-1* = *both-mono-ordered-semiring-1* +
assumes *SN*: *SN* {(*x*,*y*) . *arc-pos y* ∧ *x* ≻ *y*}

locale *weak-SN-strict-mono-ordered-semiring-1* =
fixes *weak-gt* :: *'a* :: *ordered-semiring-1* ⇒ *'a* ⇒ *bool*
and *default* :: *'a*
and *mono* :: *'a* ⇒ *bool*
assumes *weak-gt-mono*: ∀ *x y*. (*x*,*y*) ∈ *set xys* ⟶ *weak-gt x y* ⟹ ∃ *gt*. *SN-strict-mono-ordered-semiring-1 default gt mono* ∧ (∀ *x y*. (*x*,*y*) ∈ *set xys* ⟶ *gt x y*)

locale *weak-SN-both-mono-ordered-semiring-1* =
fixes *weak-gt* :: *'a* :: *ordered-semiring-1* ⇒ *'a* ⇒ *bool*
and *default* :: *'a*
and *arc-pos* :: *'a* ⇒ *bool*
assumes *weak-gt-both-mono*: ∀ *x y*. (*x*,*y*) ∈ *set xys* ⟶ *weak-gt x y* ⟹ ∃ *gt*. *SN-both-mono-ordered-semiring-1 default gt arc-pos* ∧ (∀ *x y*. (*x*,*y*) ∈ *set xys* ⟶ *gt x y*)

class *poly-carrier* = *ordered-semiring-1* + *comm-semiring-1*

locale *poly-order-carrier* = *SN-one-mono-ordered-semiring-1 default gt*
for *default* :: *'a* :: *poly-carrier* **and** *gt* (**infix** ≻ *50*) +
fixes *power-mono* :: *bool*
and *discrete* :: *bool*
assumes *times-gt-mono*: ⟦*y* ≻ *z*; *x* ≥ *1*⟧ ⟹ *y* ∗ *x* ≻ *z* ∗ *x*
and *power-mono*: *power-mono* ⟹ *x* ≻ *y* ⟹ *y* ≥ *0* ⟹ *n* ≥ *1* ⟹ *x* ^ *n* ≻ *y* ^ *n*
and *discrete*: *discrete* ⟹ *x* ≥ *y* ⟹ ∃ *k*. *x* = (((+) *1*)^^*k*) *y*

class *large-ordered-semiring-1* = *poly-carrier* +
assumes *ex-large-of-nat*: ∃ *x*. *of-nat x* ≥ *y*

context *ordered-semiring-1*
begin
lemma *pow-mono*: **assumes** *ab*: $a \geq b$ **and** *b*: $b \geq 0$
shows $a \mathbin{\hat{}} n \geq b \mathbin{\hat{}} n \wedge b \mathbin{\hat{}} n \geq 0$
⟨*proof*⟩

lemma *pow-ge-zero*[*intro*]: **assumes** *a*: $a \geq (0 :: 'a)$
shows $a \mathbin{\hat{}} n \geq 0$
⟨*proof*⟩
end

lemma *of-nat-ge-zero*[*intro*,*simp*]: *of-nat* $n \geq (0 :: 'a :: ordered\text{-}semiring\text{-}1)$
⟨*proof*⟩

lemma *mult-ge-zero*[*intro*]: $(a :: 'a :: ordered\text{-}semiring\text{-}1) \geq 0 \Longrightarrow b \geq 0 \Longrightarrow a * b \geq 0$
⟨*proof*⟩

lemma *pow-mono-one*: **assumes** *a*: $a \geq (1 :: 'a :: ordered\text{-}semiring\text{-}1)$
shows $a \mathbin{\hat{}} n \geq 1$
⟨*proof*⟩

lemma *pow-mono-exp*: **assumes** *a*: $a \geq (1 :: 'a :: ordered\text{-}semiring\text{-}1)$
shows $n \geq m \Longrightarrow a \mathbin{\hat{}} n \geq a \mathbin{\hat{}} m$
⟨*proof*⟩

lemma *mult-ge-one*[*intro*]: **assumes** *a*: $(a :: 'a :: ordered\text{-}semiring\text{-}1) \geq 1$
and *b*: $b \geq 1$
shows $a * b \geq 1$
⟨*proof*⟩

lemma *sum-list-ge-mono*: **fixes** *as* :: ('*a* :: *ordered-semiring-0*) *list*
assumes *length as* = *length bs*
and $\bigwedge i.\ i <$ *length bs* $\Longrightarrow$ *as* ! $i \geq$ *bs* ! i
shows *sum-list as* $\geq$ *sum-list bs*
⟨*proof*⟩

lemma *sum-list-ge-0-nth*: **fixes** *xs* :: ('*a* :: *ordered-semiring-0*)*list*
assumes *ge*: $\bigwedge i.\ i <$ *length xs* $\Longrightarrow$ *xs* ! $i \geq 0$
shows *sum-list xs* ≥ 0
⟨*proof*⟩

lemma *sum-list-ge-0*: **fixes** *xs* :: ('*a* :: *ordered-semiring-0*)*list*
assumes *ge*: $\bigwedge x.\ x \in$ *set xs* $\Longrightarrow x \geq 0$
shows *sum-list xs* ≥ 0
⟨*proof*⟩

lemma *foldr-max*: $a \in$ *set as* $\Longrightarrow$ *foldr max as b* $\geq (a :: 'a :: ordered\text{-}ab\text{-}semigroup)$

⟨*proof*⟩

lemma *of-nat-mono*[*intro*]: **assumes** $n \geq m$ **shows** (*of-nat* n :: *'a* :: *ordered-semiring-1*) $\geq$ *of-nat* m
⟨*proof*⟩

non infinitesmal is the same as in the CADE07 bounded increase paper

definition *non-inf* :: *'a rel* ⇒ *bool*
where *non-inf* $r \equiv \forall\ a\ f.\ \exists\ i.\ (f\ i,\ f\ (Suc\ i)) \notin r \lor (f\ i,\ a) \notin r$

lemma *non-infI*[*intro*]: **assumes** $\bigwedge a\ f.\ [\![\bigwedge i.\ (f\ i,\ f\ (Suc\ i)) \in r]\!] \Longrightarrow \exists\ i.\ (f\ i,\ a) \notin r$
shows *non-inf* r
⟨*proof*⟩

lemma *non-infE*[*elim*]: **assumes** *non-inf* r **and** $\bigwedge i.\ (f\ i,\ f\ (Suc\ i)) \notin r \lor (f\ i,\ a) \notin r \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *non-inf-image*:
assumes *ni*: *non-inf* r **and** *image*: $\bigwedge a\ b.\ (a,b) \in s \Longrightarrow (f\ a,\ f\ b) \in r$
shows *non-inf* s
⟨*proof*⟩

lemma *SN-imp-non-inf*: *SN* $r \Longrightarrow$ *non-inf* r
⟨*proof*⟩

lemma *non-inf-imp-SN-bound*: *non-inf* $r \Longrightarrow$ *SN* $\{(a,b).\ (b,c) \in r \land (a,b) \in r\}$
⟨*proof*⟩

end

5 Carriers of Strongly Normalizing Orders

theory *SN-Order-Carrier*
imports
SN-Orders
HOL.Rat
begin

This theory shows that standard semirings can be used in combination with polynomials, e.g. the naturals, integers, and arbitrary Archemedean fields by using delta-orders.

It also contains the arctic integers and arctic delta-orders where 0 is -infty, 1 is zero, + is max and * is plus.

5.1 The standard semiring over the naturals

instantiation *nat* :: *large-ordered-semiring-1*
begin
instance ⟨*proof*⟩
end

definition *nat-mono* :: *nat* ⇒ *bool* **where** *nat-mono* $x \equiv x \neq 0$

interpretation *nat-SN*: *SN-strict-mono-ordered-semiring-1 1* (>) :: *nat* ⇒ *nat* ⇒ *bool nat-mono*
⟨*proof*⟩

interpretation *nat-poly*: *poly-order-carrier 1* (>) :: *nat* ⇒ *nat* ⇒ *bool True discrete*
⟨*proof*⟩

5.2 The standard semiring over the Archimedean fields using delta-orderings

definition *delta-gt* :: 'a :: *floor-ceiling* ⇒ 'a ⇒ 'a ⇒ *bool* **where**
delta-gt $\delta \equiv (\lambda\ x\ y.\ x - y \geq \delta)$

lemma *non-inf-delta-gt*: **assumes** *delta*: $\delta > 0$
shows *non-inf* {(a,b) . *delta-gt* δ a b} (**is** *non-inf ?r*)
⟨*proof*⟩

lemma *delta-gt-SN*: **assumes** *dpos*: $\delta > 0$ **shows** *SN* {(x,y). $0 \leq y \wedge$ *delta-gt* δ x y}
⟨*proof*⟩

definition *delta-mono* :: 'a :: *floor-ceiling* ⇒ *bool* **where** *delta-mono* $x \equiv x \geq 1$

subclass (**in** *floor-ceiling*) *large-ordered-semiring-1*
⟨*proof*⟩

lemma *delta-interpretation*: **assumes** *dpos*: $\delta > 0$ **and** *default*: $\delta \leq$ *def*
shows *SN-strict-mono-ordered-semiring-1 def* (*delta-gt* δ) *delta-mono*
⟨*proof*⟩

lemma *delta-poly*: **assumes** *dpos*: $\delta > 0$ **and** *default*: $\delta \leq$ *def*
shows *poly-order-carrier def* (*delta-gt* δ) ($1 \leq \delta$) *False*
⟨*proof*⟩

lemma *delta-minimal-delta*: **assumes** $\bigwedge x\ y.\ (x,y) \in$ *set xys* $\Longrightarrow x > y$
shows $\exists\ \delta > 0.\ \forall\ x\ y.\ (x,y) \in$ *set xys* $\longrightarrow$ *delta-gt* δ x y
⟨*proof*⟩

interpretation *weak-delta-SN*: *weak-SN-strict-mono-ordered-semiring-1* (>) *1 delta-mono*
⟨*proof*⟩

5.3 The standard semiring over the integers

definition *int-mono* :: *int* ⇒ *bool* **where** *int-mono* $x \equiv x \geq 1$

instantiation *int* :: *large-ordered-semiring-1*
begin
instance
⟨*proof*⟩
end

lemma *non-inf-int-gt*: *non-inf* {(*a*,*b* :: *int*) . $a > b$} (**is** *non-inf ?r*)
⟨*proof*⟩

interpretation *int-SN*: *SN-strict-mono-ordered-semiring-1 1* (>) :: *int* ⇒ *int* ⇒ *bool int-mono*
⟨*proof*⟩

interpretation *int-poly*: *poly-order-carrier 1* (>) :: *int* ⇒ *int* ⇒ *bool True discrete*
⟨*proof*⟩

5.4 The arctic semiring over the integers

plus is interpreted as max, times is interpreted as plus, 0 is -infinity, 1 is 0

datatype *arctic* = *MinInfty* | *Num-arc int*

instantiation *arctic* :: *ord*
begin
fun *less-eq-arctic* :: *arctic* ⇒ *arctic* ⇒ *bool* **where**
less-eq-arctic MinInfty x = *True*
| *less-eq-arctic* (*Num-arc* -) *MinInfty* = *False*
| *less-eq-arctic* (*Num-arc y*) (*Num-arc x*) = ($y \leq x$)

fun *less-arctic* :: *arctic* ⇒ *arctic* ⇒ *bool* **where**
less-arctic MinInfty x = *True*
| *less-arctic* (*Num-arc* -) *MinInfty* = *False*
| *less-arctic* (*Num-arc y*) (*Num-arc x*) = ($y < x$)

instance ⟨*proof*⟩
end

instantiation *arctic* :: *ordered-semiring-1*
begin
fun *plus-arctic* :: *arctic* ⇒ *arctic* ⇒ *arctic* **where**
plus-arctic MinInfty y = *y*
| *plus-arctic x MinInfty* = *x*

| *plus-arctic* (*Num-arc x*) (*Num-arc y*) = (*Num-arc* (*max x y*))

fun *times-arctic* :: *arctic* ⇒ *arctic* ⇒ *arctic* **where**
times-arctic MinInfty y = *MinInfty*
| *times-arctic x MinInfty* = *MinInfty*
| *times-arctic* (*Num-arc x*) (*Num-arc y*) = (*Num-arc* (*x* + *y*))

definition *zero-arctic* :: *arctic* **where**
zero-arctic = *MinInfty*

definition *one-arctic* :: *arctic* **where**
one-arctic = *Num-arc 0*

instance
⟨*proof*⟩
end

fun *get-arctic-num* :: *arctic* ⇒ *int*
where *get-arctic-num* (*Num-arc n*) = *n*

fun *pos-arctic* :: *arctic* ⇒ *bool*
where *pos-arctic MinInfty* = *False*
| *pos-arctic* (*Num-arc n*) = (*0* <= *n*)

interpretation *arctic-SN*: *SN-both-mono-ordered-semiring-1 1* (>) *pos-arctic*
⟨*proof*⟩

5.5 The arctic semiring over an arbitrary archimedean field

completely analogous to the integers, where one has to use delta-orderings

datatype *'a arctic-delta* = *MinInfty-delta* | *Num-arc-delta 'a*

instantiation *arctic-delta* :: (*ord*) *ord*
begin
fun *less-eq-arctic-delta* :: *'a arctic-delta* ⇒ *'a arctic-delta* ⇒ *bool* **where**
less-eq-arctic-delta MinInfty-delta x = *True*
| *less-eq-arctic-delta* (*Num-arc-delta -*) *MinInfty-delta* = *False*
| *less-eq-arctic-delta* (*Num-arc-delta y*) (*Num-arc-delta x*) = (*y* ≤ *x*)

fun *less-arctic-delta* :: *'a arctic-delta* ⇒ *'a arctic-delta* ⇒ *bool* **where**
less-arctic-delta MinInfty-delta x = *True*
| *less-arctic-delta* (*Num-arc-delta -*) *MinInfty-delta* = *False*
| *less-arctic-delta* (*Num-arc-delta y*) (*Num-arc-delta x*) = (*y* < *x*)

instance ⟨*proof*⟩
end

instantiation *arctic-delta* :: (*linordered-field*) *ordered-semiring-1*
begin
fun *plus-arctic-delta* :: *'a arctic-delta* ⇒ *'a arctic-delta* ⇒ *'a arctic-delta* **where**
plus-arctic-delta MinInfty-delta y = *y*
| *plus-arctic-delta x MinInfty-delta* = *x*
| *plus-arctic-delta* (*Num-arc-delta x*) (*Num-arc-delta y*) = (*Num-arc-delta* (*max x y*))

fun *times-arctic-delta* :: *'a arctic-delta* ⇒ *'a arctic-delta* ⇒ *'a arctic-delta* **where**
times-arctic-delta MinInfty-delta y = *MinInfty-delta*
| *times-arctic-delta x MinInfty-delta* = *MinInfty-delta*
| *times-arctic-delta* (*Num-arc-delta x*) (*Num-arc-delta y*) = (*Num-arc-delta* (*x* + *y*))

definition *zero-arctic-delta* :: *'a arctic-delta* **where**
zero-arctic-delta = *MinInfty-delta*

definition *one-arctic-delta* :: *'a arctic-delta* **where**
one-arctic-delta = *Num-arc-delta 0*

instance
⟨*proof*⟩
end

x >d y is interpreted as y = -inf or (x,y != -inf and x >d y)

fun *gt-arctic-delta* :: *'a* :: *floor-ceiling* ⇒ *'a arctic-delta* ⇒ *'a arctic-delta* ⇒ *bool*
where *gt-arctic-delta* δ *- MinInfty-delta* = *True*
| *gt-arctic-delta* δ *MinInfty-delta* (*Num-arc-delta -*) = *False*
| *gt-arctic-delta* δ (*Num-arc-delta x*) (*Num-arc-delta y*) = *delta-gt* δ *x y*

fun *get-arctic-delta-num* :: *'a arctic-delta* ⇒ *'a*
where *get-arctic-delta-num* (*Num-arc-delta n*) = *n*

fun *pos-arctic-delta* :: (*'a* :: *floor-ceiling*) *arctic-delta* ⇒ *bool*
where *pos-arctic-delta MinInfty-delta* = *False*
| *pos-arctic-delta* (*Num-arc-delta n*) = ($0 \le n$)

lemma *arctic-delta-interpretation*: **assumes** *dpos*: $\delta > 0$ **shows** *SN-both-mono-ordered-semiring-1 1* (*gt-arctic-delta* δ) *pos-arctic-delta*
⟨*proof*⟩

fun *weak-gt-arctic-delta* :: (*'a* :: *floor-ceiling*) *arctic-delta* ⇒ *'a arctic-delta* ⇒ *bool*
where *weak-gt-arctic-delta - MinInfty-delta* = *True*
| *weak-gt-arctic-delta MinInfty-delta* (*Num-arc-delta -*) = *False*
| *weak-gt-arctic-delta* (*Num-arc-delta x*) (*Num-arc-delta y*) = ($x > y$)

interpretation *weak-arctic-delta-SN*: *weak-SN-both-mono-ordered-semiring-1 weak-gt-arctic-delta 1 pos-arctic-delta*

⟨*proof*⟩

end

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, Aug. 1999.

[2] C. Sternagel. *Automatic Certification of Termination Proofs*. PhD thesis, University of Innsbruck, Institute of Computer Science, 2010. not finished yet.

[3] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2009*, pages 452–468. Springer, 2009.